UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Desarrollo de una aplicación para un terminal móvil con soporte para geolocalización

Autor: Luis Recuenco Pérez

Tutores: Jorge Blasco Alís, José María de Fuentes García-Romero de Tejada

Octubre 2010

Agradecimientos

En este justo momento, no puedo evitar acordarme del verano de 2003. Terminé mi primer año de Universidad, con resultados tan poco satisfactorios como para hacerme creer que esta carrera no había sido la mejor opción. No estaba a la altura de la misma. Hoy, miro atrás y me alegro de la decisión que tomé de seguir adelante. No podría haber elegido mejor.

Ha sido un camino duro, largo, con momentos buenos y malos, pero han sido ocho años de momentos inolvidables donde no he estado sólo.

Agradecer todo el apoyo de mi familia, en especial el de mis padres y el de mi abuela Esperanza quien, a pesar de su edad, siempre tenía ese hueco para preguntarme y animarme a no dejar nunca la carrera y en particular este proyecto que, en algún momento, pensé que jamás llegaría a terminar.

Me gustaría mencionar de forma muy especial a Irene. Muchas gracias por haber compartido conmigo estos últimos años de Universidad y hacer que se hayan convertido en los mejores de mi vida por el simple hecho de estar conmigo. Gracias por tener tanta paciencia en los momentos difíciles y, por último, y más importante, gracias por quererme.

Andrés, Sergio, Saúl. Gracias por estar siempre ahí. Gracias Sergio por el diseño del icono y la pantalla principal de la aplicación.

No puedo olvidarme de mis amigos de Universidad, Viñu, Javo, Kiko, Miki, Natalia, Rosa, Raúl, Vaíllo, Manu, Carlos, Fer, Rober, Santi (espero no dejarme a ninguno). Gracias a todos por hacerme menos duros estos años y por todos los geniales momentos que hemos pasado dentro y fuera de la Universidad.

En especial, mencionar a Javo y Viñu. Me llevo en vosotros dos unos verdaderos amigos, gracias.

Por último mencionar a mis compañeros de la empresa GotFeeling?. Gracias Laura por esas pequeñas ayudas de diseño para mejorar gráficamente la aplicación.

Gracias Jorge, Chema, por estar siempre ahí cuando lo he necesitado y por guiarme de la mejor manera en toda la realización de este Proyecto. He aprendido mucho de vosotros. No podía haber tenido mejores tutores.

Gracias a todos.

Aquí termina una etapa. Empieza mi nueva vida.

Índice general

Índice de figuras XII

Índice de tablas XV

Índice de requisitos XVII

Índice de pruebas XVIII

Capítulo 1: Introducción 19

1.1 Motivación del proyecto 19

1.2 Objetivos 21

1.3 Contenido de la memoria 22

Capítulo 2: Análisis 24

2.1 Introducción 24

2.2 Servicios basados en localización 24

2.2.1 Definición de LBS 24

2.2.2 Componentes de un sistema LBS 25

2.2.3 Búsqueda de información en un sistema LBS 27

2.2.4 Aplicaciones de LBS 28

2.2.5 Técnicas de localización 29

2.2.5.1 Geolocalización mediante GPS (General Positioning System) 29

2.2.5.2 Geolocalización mediante WPS (WiFi Positioning System) 30

2.2.5.3 Geolocalización mediante torres de telefonía (Cell ID) 30

2.3 Comparación de SmartPhones 33

2.3.1 Análisis económico *33*

2.3.2 Análisis general entre iPhone y Android *37*

2.3.2.1 Desarrollo *37*

2.3.2.2 Pago al desarrollador *37*

2.3.2.3 Visibilidad *38*

2.3.2.4 Modelo de plataforma *38*

2.3.2.5 Fragmentación *38*

2.3.3 Elección de la plataforma. Conclusión final. *39*

2.4 Aplicaciones similares en la plataforma **39**

2.4.1 Google Maps *39*

2.4.2 AroundMe *41*

2.4.3 Comparación de la aplicación a desarrollar. *42*

2.5 Análisis de tecnologías: KML, KMZ, XML y JSON **43**

2.5.1 KML y KMZ *43*

2.5.2 XML y JSON *45*

2.6 Fuentes de datos utilizadas **47**

2.6.1 Obtención de la localización del usuario *47*

2.6.2 API para obtener lugares cercanos *47*

2.6.2.1 Google Maps *47*

2.6.2.2 Mi nube *49*

2.6.2.3 Qype y 11870 *51*

2.6.3 API para fotos, vídeos y opiniones *53*

2.6.3.1 Google Maps *53*

2.6.3.1.1 Utilización de las API proporcionadas por Google Maps *53*

2.6.3.1.2 Utilización de la información de la web de Google Maps *54*

2.6.3.2 Minube *55*

2.6.3.3 Qype *56*

2.6.4 Elección final *57*

2.6.5 Obtención de las rutas *58*

2.6.6 Redes sociales *60*

2.6.6.1 Facebook *61*

2.6.6.2 Twitter *62*

2.7 Diagrama de casos de uso **62**

2.8 Catálogo de requisitos software **64**

2.8.1 Formato de la especificación de requisitos *64*

2.8.2 Requisitos funcionales *64*

2.8.3 Requisitos no funcionales *70*

2.9 Plan de pruebas **72**

2.9.1 Formato del catálogo de pruebas *72*

2.9.2 Catálogo de pruebas de aceptación *73*

2.9.3 Matriz de pruebas-requisitos *75*

Capítulo 3: Diseño **77**

3.1 Diseño del software **77**

3.1.1 Diseño de la arquitectura del sistema *77*

3.1.2 Diseño del controlador de vistas principales *79*

3.1.3 Diseño del controlador de rutas *88*

3.1.4 Diseño del controlador del lugar turístico *91*

3.2 Diagramas de secuencia **98**

3.2.1 Inicio de la aplicación *98*

3.2.2 Búsqueda de lugares cualesquiera cercanos 99

3.2.3 Ver un lugar y búsqueda de fotos y opiniones 100

3.2.4 Visualización de fotos 101

3.2.5 Visualización de opiniones 101

3.2.6 Cálculo de rutas 102

3.2.7 Compartir en Facebook 103

3.2.8 Guardar favorito 104

Capítulo 4: Implementación 106

4.1 Aspectos de la implementación 106

4.1.1 Localización del usuario 106

4.1.1.1 Modo de operación 106

4.1.1.2 Precisión 107

4.1.1.3 Resolución 107

4.1.1.4 Puesta en marcha y apagado de la localización 108

4.1.1.5 Geocodificación inversa 108

4.1.2 Autenticación básica en Twitter 109

4.1.3 Gestión de las imágenes 111

4.1.4 Framework Three20 114

4.1.5 Envío de favoritos por Bluetooth 115

4.1.6 Caracteres especiales HTML 119

4.1.7 Google Mobilizer 120

4.2 Resultados del plan de pruebas 121

Capítulo 5: Gestión del proyecto 122

5.1 Planificación del proyecto 122

5.1.1 Planificación inicial *122*

5.1.2 Planificación real *124*

5.2 Medios técnicos empleados **125**

5.2.1 Hardware *126*

5.2.2 Software *126*

5.3 Análisis económico **127**

5.3.1 Metodología de estimación de costes *127*

5.3.2 Análisis de costes estimados *128*

5.3.2.1 Estimación del coste de personal *128*

5.3.2.2 Estimación del coste del Hardware *129*

5.3.2.3 Estimación del coste del Software *129*

5.3.2.4 Estimación de los costes indirectos *130*

5.3.2.5 Estimación de los costes totales *131*

5.3.3 Análisis de costes reales *131*

5.3.3.1 Coste real de personal *132*

5.3.3.2 Coste real del Hardware *132*

5.3.3.3 Coste real del Software *133*

5.3.3.4 Costes indirectos reales *134*

5.3.3.5 Costes totales reales *134*

5.4 Análisis de la forma de venta de la aplicación **135**

5.4.1 Venta ordinaria no gratuita *136*

5.4.2 Venta a través del sistema publicitario iAd *138*

5.4.2.1 Definición de iAd *138*

5.4.2.2 Análisis económico de iAd *139*

Capítulo 6: Conclusiones y líneas futuras **143**

6.1 Conclusiones sobre el proyecto 143
6.2 Conclusiones a nivel personal 144
6.3 Líneas futuras 145
6.3.1 Adecuación al nuevo hardware y software *145*
6.3.1.1 iPhone 4 *145*
6.3.1.2 iOS 4 *145*
6.3.2 Realidad aumentada *146*
6.3.3 iPad/Android *148*
6.3.4 iAd *149*
Anexo A: Manual de la aplicación 150
A.1 Explicación de las pantallas principales 150
A.1.1 Introducción y barra superior *150*
A.1.2 Pestañas Favoritos y Preferencias *154*
A.1.3 Pestañas Lista y Mapa *157*
A.1.4 Pantalla principal *159*
A.1.4.1 Posición en el mapa *160*
A.1.4.2 Imágenes *161*
A.1.4.3 Opiniones *162*
A.1.4.4 Vídeos *163*
A.1.4.5 Otras opciones *164*
A.2 Usos típicos de la aplicación 167
A.2.1 Búsqueda de lugares cercanos *167*
A.2.1.1 Lugares turísticos *167*
A.2.1.2 Lugares no turísticos *172*

A.2.2 Planificación de un viaje *174*

Anexo B: Backup y control de versiones **177**

B.1 Backup - Dropbox **177**

B.2 Control de versiones - Git **179**

B.2.1 Descripción del servicio *179*

B.2.2 Configuración del control de versiones *179*

Fuentes de las imágenes utilizadas **183**

Bibliografía **184**

Índice de figuras

Figura 1: Martin Cooper junto con el primer teléfono móvil 19
Figura 2: Comparación de los terminales móviles 20
Figura 3: LBS como intersección de tecnologías 25
Figura 4: Actores de LBS 26
Figura 5: Funcionamiento de LBS 27
Figura 6: Usos de LBS 28
Figura 7: Localización mediante GPS 29
Figura 8: Localización mediante WPS 30
Figura 9: Localización mediante torres de telefonía 31
Figura 10: Comparación de métodos LBS 32
Figura 11: Aumento de aplicaciones LBS 33
Figura 12: Número de aplicaciones según compañías 34
Figura 13: Porcentaje de aplicaciones gratis y de pago 35
Figura 14: Número de aplicaciones descargadas al mes 36
Figura 15: Porcentaje de usuarios que compran una aplicación de pago al mes 36
Figura 16: Google Maps en iPhone (I) 40
Figura 17: Google Maps en iPhone (II) 40
Figura 18: AroundMe en iPhone (I) 41
Figura 19: AroundMe en iPhone (II) 42
Figura 20: Ejemplo de fichero KML 44
Figura 21: Ejemplo de fichero JSON 46
Figura 22: JSON de lugares Google Maps 49
Figura 23: JSON de lugares Minube 51
Figura 24: JSON de lugares Qype 52
Figura 25: JSON de fotos en Google Maps 53
Figura 26: JSON de fotos Minube 56
Figura 27: JSON de opiniones Minube 56
Figura 28: JSON de opiniones Qype 57
Figura 29: JSON de rutas CloudMade 60
Figura 30: Aplicación iTourist en Facebook 61
Figura 31: Casos de uso 63
Figura 32: Diseño arquitectónico 78
Figura 33: Clases participantes en el inicio de la aplicación 79

Figura 34: Diseño del controlador de vistas principales 83
Figura 35: Diseño del controlador de rutas 88
Figura 36: Diseño del controlador del lugar turístico 92
Figura 37: Hilos de carga de información multimedia 97
Figura 38: Diagrama de secuencia - inicio de la aplicación 98
Figura 39: Diagrama de secuencia - búsqueda de lugares cercanos 99
Figura 40: Diagrama de secuencia - ver un lugar 100
Figura 41: Diagrama de secuencia - visualización de fotos 101
Figura 42: Diagrama de secuencia - visualización de opiniones 101
Figura 43: Diagrama de secuencia - cálculo de rutas 103
Figura 44: Diagrama de secuencia - compartir en Facebook 104
Figura 45: Diagrama de secuencia - guardar favoritos 105
Figura 46: Búsqueda de dispositivos Bluetooth 116
Figura 47: Google Mobilizer 121
Figura 48: Planificación inicial 124
Figura 49: Planificación real 125
Figura 50: Multicanalidad de la App Store 135
Figura 51: Número de aplicaciones a vender para ROI = 30% 136
Figura 52: Número de días para ROI = 30% (CPM + CPC) 140
Figura 53: Número de días para ROI = 30% (CPM) 141
Figura 54: Comparación de CPM + CPC frente a CPM 142
Figura 55: Multitarea en iOS 4 146
Figura 56: Ejemplo de realidad aumentada 147
Figura 57: Realidad aumentada aplicada al proyecto 148
Figura 58: Aplicación para iPad y Android 149
Figura 59: iAd 149
Figura 60: a) "Splash" b) Elección de idioma c)Información 150
Figura 61: Lista principal de lugares 151
Figura 62: Ningún lugar 151
Figura 63: Explicación barra superior 152
Figura 64: **Aspecto de la aplicación** *a) Tras pulsar 2 b) Tras pulsar 3* 153
Figura 65: **Aspecto de la aplicación** *a) Tras pulsar 4 b) Tras pulsar 5* 153
Figura 66: Explicación barra inferior 154
Figura 67: a) Favoritos b) Preferencias 154
Figura 68: Explicación barra superior favoritos 155
Figura 69: Envío de favoritos por bluetooth 156
Figura 70: a) Borrar favoritos b) Ningún favorito 156

Figura 71: a) Lista b) Mapa *157*
Figura 72: Barra superior mapa *158*
Figura 73: Pantalla principal de un lugar *159*
Figura 74: Mapa del lugar *160*
Figura 75: Fotos del lugar *161*
Figura 76: Opiniones del lugar *162*
Figura 77: Vídeos del lugar *163*
Figura 78: Barra superior pantalla principal de un lugar *164*
Figura 79: Llamar o agregar lugar *164*
Figura 80: Recomendar lugar (I) *165*
Figura 81: Recomendar lugar (II) *166*
Figura 82: Lugares turísticos cercanos *167*
Figura 83: Ruta desde la posición actual *168*
Figura 84: Ruta en la aplicación "Mapas" *169*
Figura 85: Barra superior ruta (I) *170*
Figura 86: Ruta con varios lugares *170*
Figura 87: Indicaciones paso a paso y vista satélite *171*
Figura 88: Barra superior ruta (II) *171*
Figura 89: Ruta en Google Maps *172*
Figura 90: Búsqueda de lugares no turísticos *173*
Figura 91: Búsqueda de hotel *174*
Figura 92: Elección de hotel *175*
Figura 93: Ruta del viaje planificado *176*
Figura 94: Dropbox multiplataforma *178*
Figura 95: Sincronización entre casa y el trabajo *181*
Figura 96: Gestión del repositorio Git *182*

Índice de tablas

Tabla 1: Parámetros API Google Maps 48
Tabla 2: API Google Maps para fotos y vídeos 53
Tabla 3: Direcciones Web para obtener fotos, vídeos y opiniones 54
Tabla 4: Matriz de pruebas-requisitos 76
Tabla 5: Clase PFCAppDelegate 80
Tabla 6: Clase LanguagePickerViewController 81
Tabla 7: Clase InfoViewController 82
Tabla 8: Clase ReviewsBrowserViewController 82
Tabla 9: Métodos y atributos comunes (I) 84
Tabla 10: Métodos y atributos comunes (II) 85
Tabla 11: Clase ListPlacesViewController 85
Tabla 12: Clase MapPlacesViewController 86
Tabla 13: Clase FavouritePlacesViewController 87
Tabla 14: Clase SettingsViewController 87
Tabla 15: Clase MapRouteViewController 90
Tabla 16: Clase GoogleMapsRouteViewController 91
Tabla 17: Clase GoogleSearchParser 93
Tabla 18: Clase TouristicPlace 94
Tabla 19: Clase TouristPlaceViewController 95
Tabla 20: Clase TouristPlaceVideosViewController 95
Tabla 21: Clase PhotoTestController 95
Tabla 22: Clase UserReviewsViewController 96
Tabla 23: Codificación ISO-8859-1 119
Tabla 24: Dispositivos hardware utilizados 126
Tabla 25: Herramientas de software utilizadas 127
Tabla 26: Coste de personal estimado 128
Tabla 27: Coste de hardware estimado 129
Tabla 28: Coste de software estimado 130
Tabla 29: Costes indirectos estimados 131
Tabla 30: Costes totales estimados 131
Tabla 31: Horas a tiempo completo y parcial 132
Tabla 32: Coste de personal real 132
Tabla 33: Coste de hardware real 133
Tabla 34: Coste de software real 134

Tabla 35: Costes indirectos reales *134*
Tabla 36: Costes totales reales *134*
Tabla 37: Años para ROI = 30% variando precio *137*
Tabla 38: ROI variando las descargas diarias *137*
Tabla 39: Años para ROI = 30% variando descargas diarias *138*
Tabla 40: Días para ROI = 30% variando el precio CPM (CPM + CPC) *140*
Tabla 41: Días para ROI = 30% variando precio (CPM) *141*

Índice de requisitos

Requisito 1: Ubicación del terminal 64
Requisito 2: Obtención de lugares turísticos 64
Requisito 3: Obtención de lugares concretos cercanos 65
Requisito 4: Obtención de lugares concretos 65
Requisito 5: Obtención de las características básicas del lugar 65
Requisito 6: Obtención de imágenes del lugar 65
Requisito 7: Obtención de vídeos del lugar 66
Requisito 8: Obtención de opiniones del lugar 66
Requisito 9: Obtención de datos básicos de las opiniones 66
Requisito 10: Obtención de la página web de la opinión 66
Requisito 11: Ubicación de un lugar en el mapa 67
Requisito 12: Generación de ruta desde la posición actual 67
Requisito 13: Generación de rutas entre diferentes lugares 67
Requisito 14 : Almacenamiento de lugares favoritos 67
Requisito 15: Envío de favoritos por Bluetooth 68
Requisito 16: Compartir en redes sociales 68
Requisito 17: Compartir por correo electrónico 68
Requisito 18: Llamada telefónica al lugar 68
Requisito 19: Ubicación de todos los lugares en el mapa 69
Requisito 20: Ordenar lugares 69
Requisito 21: Agregar lugar como contacto 69
Requisito 22: Guardar ruta 69
Requisito 23: Indicaciones paso a paso para las rutas 70
Requisito 24: Obtención de la ruta en la aplicación de Google Maps 70
Requisito 25: Conexión a Internet 70
Requisito 26: Plataforma de desarrollo 70
Requisito 27: Vídeos dentro de la aplicación 71
Requisito 28: Base de datos de Google Maps 71
Requisito 29: Interfaz de usuario 71
Requisito 30: Líneas de diseño 71
Requisito 31: Traducción a inglés 72
Requisito 32: Compatibilidad de dispositivos 72
Requisito 33: Pruebas de aceptación 72

Índice de pruebas

Prueba 1: Comprobación de lugares turísticos *73*
Prueba 2: Comprobación de lugares cercanos *73*
Prueba 3: Comprobación de imágenes y vídeos *73*
Prueba 4: Comprobación de las opiniones *74*
Prueba 5: Comprobación de ruta desde la posición actual *74*
Prueba 6: Comprobación de ruta entre varios lugares *74*
Prueba 7: Comprobación de conectividad bluetooth *74*
Prueba 8: Comprobación de llamada telefónica *75*
Prueba 9: Comprobación de funcionalidad de redes sociales *75*
Prueba 10: Comprobación de conectividad *75*

Capítulo 1: Introducción

1.1 Motivación del proyecto

Los teléfonos móviles han revolucionado la forma en la que se comunican las personas, la manera en la que se interactúa con el entorno y la forma en la que las personas entienden hoy en día la comunicación. Actualmente es extremadamente raro ver a una persona que no disponga de un teléfono móvil. Aunque esto no fue siempre así.

El primer teléfono móvil fue inventado hace prácticamente 37 años, cuando Motorola sacó al mercado el primer celular conocido como Motorola DynaTAC 8000X, gracias al investigador Martin Cooper. Éste se puede ver en la Figura 1 junto a su gran invento, muchos años después de efectuarse el mismo.

Figura 1: Martin Cooper junto con el primer teléfono móvil[1]

Desde 1973, la tecnología ha avanzado a un ritmo vertiginoso. Los teléfonos móviles han sufrido la evolución natural y cada vez, gracias a la nanotecnología, han ido disminuyendo su tamaño hasta límites insospechados, a la vez que aumentando las características hardware y software. Esto ha llevado a la aparición de los llamados teléfonos inteligentes (*SmartPhones*).

[1] Imagen obtenida de [I]

Figura 2: Comparación de los terminales móviles[2]

Los teléfonos inteligentes pueden ser considerados como ordenadores de bolsillo. Debido a su ultraportabilidad, el usuario puede, desde cualquier lugar en el que se encuentre, consultar el correo, comprar las entradas de cine para su película favorita o hacer una foto desde su terminal y compartirla con sus amigos en una red social. Esto viene dado por la capacidad que tienen los teléfonos inteligentes de estar siempre conectados a Internet, independientemente del lugar en el que se encuentre el usuario. Esto supuso una primera gran revolución junto con la llegada de las redes 3G y una velocidad adecuada para usos que van más allá de la simple consulta del correo electrónico.

[2] Imagen obtenida de [II] y [III]

La segunda gran revolución vino dada por un importante avance hardware que ha habido en los últimos años, y que es el origen de dicho Proyecto Fin de Carrera, esto es, la inclusión de chips GPS (*General Positioning System*) en estos terminales. Dicha característica, junto con la posibilidad de estar siempre conectado, forman una simbiosis perfecta que permite al usuario estar geoposicionado y utilizar los distintos servicios web que hacen uso de dichas características. Notar que los servicios de este tipo necesitan siempre confirmación por parte del usuario para utilizar la ubicación del mismo, por lo que la privacidad del mismo está siempre garantizada.

Son muchas las posibilidades y en las áreas en las que un teléfono inteligente puede sustituir a un ordenador personal. De hecho, han sido los grandes culpables de que las PDA (*Personal Digital Assistant*) carezcan de sentido y hayan desaparecido.

Por tanto, el objetivo inicial de este Proyecto fue realizar una aplicación para un terminal móvil que utilizara dichas características de geolocalización de los nuevos dispositivos móviles existentes en el mercado.

Tras pensar detenidamente en la aplicación óptima que pudiera hacer uso de dicha novedosa característica, se decidió realizar una aplicación con tinte turístico donde el usuario pueda acceder a distintos lugares de interés para realizar diferentes rutas desde donde se encuentre hasta los diferentes lugares. No se olvide que España es un país donde el turismo supone una fuente importante en la economía, así que disponer de una aplicación capaz de planificar un viaje o una escapada turística puede tener un valor añadido extra en nuestro país.

Existen numerosas aplicaciones de guías turísticas sobre una ciudad en concreto, con información estática almacenada en la misma aplicación. Nótese la gran diferencia con respecto al software que se desea desarrollar, donde la capacidad de localización del terminal y el hecho de que la información es accedida a través de Internet de forma dinámica otorga al usuario la posibilidad de obtener información sobre los distintos lugares turísticos que le rodean independientemente de la ciudad del mundo en la que se encuentre.

En el siguiente apartado se pueden ver algunos objetivos más específicos de la aplicación que se decidió desarrollar.

1.2 Objetivos

El objetivo general de este proyecto es desarrollar una aplicación móvil que haga uso de la capacidad de geolocalización del terminal para realizar una función turística útil de cara al usuario. A continuación se enumeran los objetivos específicos que debe cumplir dicho proyecto:

1. Obtener una serie de lugares turísticos de interés cercanos a la posición del usuario mostrando características concretas sobre los mismos (nombre, dirección, teléfono, puntuación, opiniones, imágenes...).

2. Conseguir geoposicionar los lugares mencionados arriba en un mapa. Esta representación podrá ser puntual (sólo el objeto turístico que se elija) o total, para hacerse una idea de la "telaraña turística" que rodea al usuario.

3. Establecer la ruta turística (conteniendo un número de lugares a elegir) que un usuario podría realizar partiendo desde la posición actual en la que se encuentre. Ésta será calculada para optimizar diversos parámetros como la duración o la distancia.

4. Las diferentes rutas deberán poderse guardar para posible consulta offline.

5. Las redes sociales tienen cada vez más importancia y repercusión en la sociedad actual. La aplicación debe permitir al usuario indicar en estos servicios Web 2.0 los diferentes sitios por los que va pasando, los que le gustaría conocer, o simplemente aquellos que le parecen lo suficientemente interesantes como para compartirlos con sus amigos.

6. La aplicación debe poder dar servicio en diferentes idiomas dando preferencia al inglés.

En diferentes revisiones de la fase de análisis se fueron añadiendo otros objetivos concretos entre los que destacan:

1. Gestión de lugares favoritos, incluyendo funcionalidad para transmitir los mismos a través de la tecnología bluetooth del dispositivo.

2. Extender la funcionalidad de la aplicación para que no sólo ofrezca lugares turísticos, sino la posibilidad de buscar cualquier lugar genérico.

1.3 Contenido de la memoria

La memoria está estructurada como sigue:

1. El capítulo 1 sitúa al lector en el contexto en el que se realiza este proyecto, describiendo el trabajo a desarrollar y los distintos objetivos marcados.

2. En el capítulo 2 se detalla el análisis de las distintas partes que componen el proyecto. Se tratarán los diversos métodos de localización del usuario y se compararán distintos modelos de teléfonos inteligentes como candidatos a realizar la aplicación para los mismos. Una vez elegida la plataforma, se verán algunas aplicaciones con características similares a la que se ha desarrollado, para justificar la innovación y la mejora que supone el software que se ha desarrollado. Tras dicho análisis del ecosistema móvil actual, se pasará a realizar un análisis más técnico, explicándose las diferentes API existentes para conseguir la información y plasmarla

en la aplicación así como los distintos diagramas de componentes y de casos de uso. Por último se muestra el catálogo de requisitos y de pruebas de aceptación.

3. En el capítulo 3 se explica en detalle cómo se ha llevado a cabo el diseño de la aplicación, explicando más a fondo los diferentes componentes que forman el software así como distintos diagramas que explican el comportamiento a bajo nivel del sistema.

4. El capítulo 4 trata sobre la implementación del sistema. Se detallará (a nivel de código) distintas partes importantes o peculiares del proyecto para finalmente hablar sobre la consecución de las distintas pruebas de aceptación.

5. El capítulo 5 se corresponde con la gestión del proyecto y se pueden ver los diagramas de Gantt utilizados para la planificación del tiempo así como los medios hardware y software empleados para realizar el presupuesto estimado y real del mismo. Tras dicha parte común en cualquier capítulo de gestión en un Proyecto Fin de Carrera, se realiza un análisis extenso y detallado de la forma de venta de la aplicación con el objetivo de maximizar el beneficio. Se tratarán dos formas de venta, a través del medio de distribución digital habitual cobrando un precio ajustado por la aplicación, o bien valerse de la publicidad para recaudar el importe que se desea.

6. En el capítulo 6 se detallan las conclusiones del proyecto y las distintas líneas de futuro con las que se puede continuar el desarrollo.

7. El anexo A se trata de un completo y detallado manual sobre el uso de la aplicación. Cuenta con numerosas imágenes muy descriptivas intentando clarificar las distintas pantallas de las que consta la aplicación desarrollada.

8. En el anexo B se cuenta el protocolo seguido de *backups* y control de versiones llevado en el proyecto, así como la metodología seguida para el intercambio de información entre diferentes ordenadores (en concreto el personal y el del trabajo) con los que se ha desarrollado el proyecto.

Capítulo 2: Análisis

2.1 Introducción

En este capítulo se tratará de explicar todos los pasos seguidos en el análisis del problema que ha planteado el presente Proyecto Fin de Carrera. En concreto, las distintas secciones que se analizan son:

- Servicios basados en localización
- Comparación de *SmartPhones*
- Aplicaciones similares en la plataforma
- Análisis de tecnologías: KML, KMZ, XML y JSON
- Fuentes de datos utilizadas
- Diagrama de bloques funcional
- Diagrama de casos de uso
- Catálogo de requisitos software
- Plan de pruebas

2.2 Servicios basados en localización

Para llevar a cabo el objetivo principal del proyecto, esto es, conseguir una número indefinido de lugares turísticos alrededor de la posición del usuario, es necesario que el dispositivo móvil del mismo esté localizado. Aquí es donde entran en juego los servicios basados en localización e implica el primer análisis necesario en el ámbito de la aplicación.

2.2.1 Definición de LBS

Los teléfonos móviles e Internet han revolucionado la comunicación y el estilo de vida de la mayoría de las personas. El creciente número de teléfonos móviles inteligentes (*SmartPhones*) permiten a sus usuarios acceder a Internet en cualquier lugar y cuando se desee. Tareas tan comunes como comprar una entrada en el cine más cercano hasta reservar mesa en el restaurante chino que se encuentre a menos de 2 km del usuario no serían posible sin los servicios basados en localización, también conocidos como LBS (*Location Based Systems*). Por tanto, los sistemas LBS consisten en servicios que se basan, en la mayoría de los casos, en la información de la ubicación geográfica del usuario.

Los sistemas LBS pueden ser vistos como la intersección de tres tecnologías: los sistemas de información geográfica o GIS (*Geographical Information Systems*), Internet, y los dispositivos móviles (ver Figura 3).

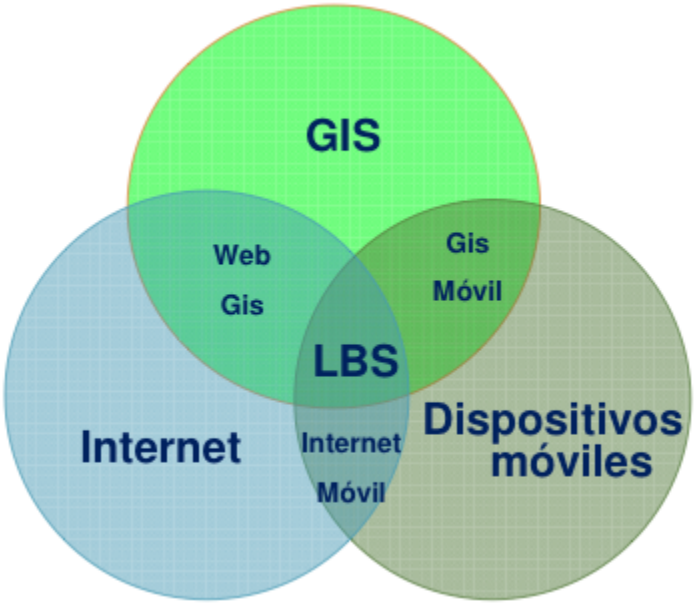

Figura 3: LBS como intersección de tecnologías[3]

Los sistemas LBS y GIS, aunque relacionados, tienen diferentes orígenes y grupos potenciales de usuarios completamente diferentes. Los sistemas GIS fueros desarrollados durante décadas teniendo como objetivo final las aplicaciones geográficas de carácter profesional. Los sistemas LBS son de reciente aparición debido a la evolución de los servicios móviles. En relación a los usuarios potenciales de ambos servicios, GIS tiene como fin satisfacer las necesidades de usuarios avanzados con necesidades muy amplias y específicas. Además, los sistemas GIS requieren en su gran mayoría grandes y potentes sistemas de computación. LBS por el contrario, está diseñado para grupos de usuarios no profesionales y con restricciones de hardware muy acentuadas, como son los sistemas terminales móviles con una batería, vida de uso y pantallas limitadas.

2.2.2 Componentes de un sistema LBS

Para que un sistema LBS sea capaz de funcionar, se necesita una infraestructura y unos elementos necesarios. Éstos son:

- **Dispositivo móvil:** El elemento con el que interaccionará el usuario y a través del cual se pedirá la información necesaria. Dicho dispositivo móvil puede tratarse de una PDA, un teléfono móvil (el más común), un ordenador portátil o incluso un navegador de un coche.

[3] Imagen obtenida de [IV]

- **Red de comunicaciones:** El segundo componente es la red móvil, la cual transfiere la información del usuario y la petición del servicio desde el terminal móvil hasta el proveedor de servicios y entrega finalmente el resultado de la petición al usuario.

- **Componente de posicionamiento:** Para poder procesar la petición del usuario, el sistema necesita la información de posicionamiento del usuario. Ésta puede obtenerse a través de diferentes técnicas (se hablará más en profundidad de esto más adelante). Las más comunes son la utilización de GPS, la red de comunicaciones móvil a la que está conectado el dispositivo (WiFi) o a través de las torres de telefonía móvil.

- **Proveedor de servicios:** Ofrece un número diferente de servicios al usuario y es el responsable de procesar la petición del servicio hecha por el usuario. Tales servicios pueden ser calcular la ruta desde la posición del usuario hasta el cine más cercano, obtener el tiempo que se espera en la posición actual en las próximas cinco horas, etc.

- **Proveedor de contenidos:** Los proveedores de servicio, por regla general, no almacenan y mantienen todo el tipo de información que puede ser requerida por los usuarios. Aquí es donde entra en juego el proveedor de contenidos para realizar dichas tareas que quedan fuera del ámbito del proveedor de servicios.

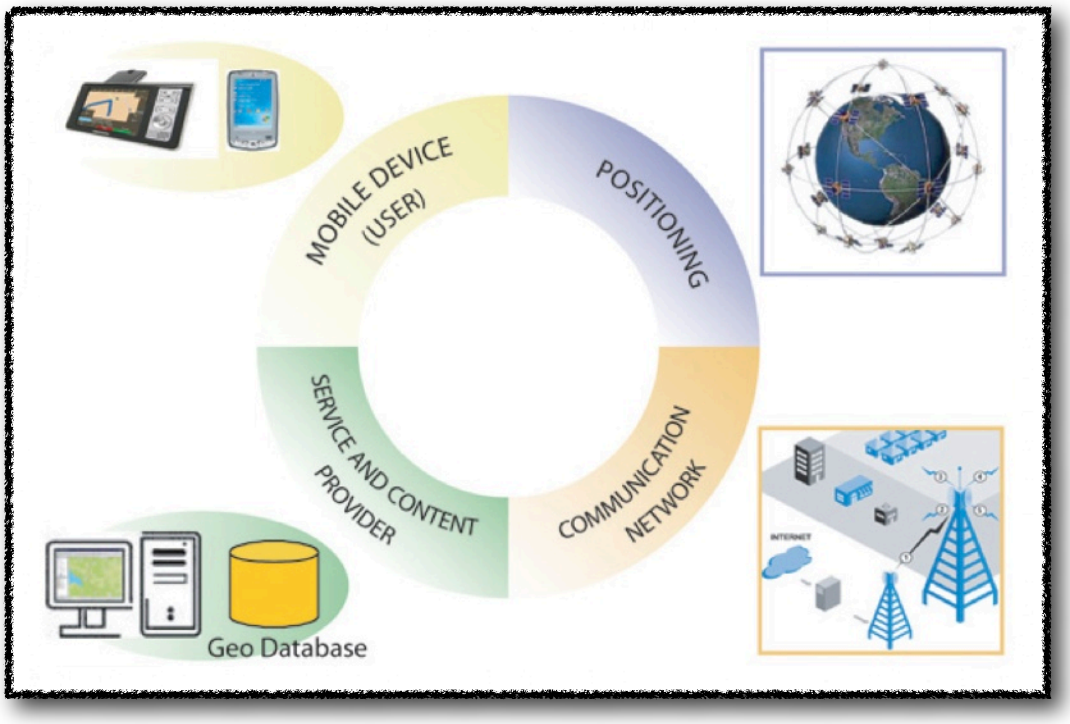

Figura 4: Actores de LBS[4]

[4] Imagen obtenida de [IV]

2.2.3 Búsqueda de información en un sistema LBS

Para intentar clarificar el flujo de información existente en un sistema LBS, se pondrá como ejemplo que el usuario necesita obtener una serie de lugares turísticos cercanos a su posición actual. El flujo se puede ver en la Figura 5:

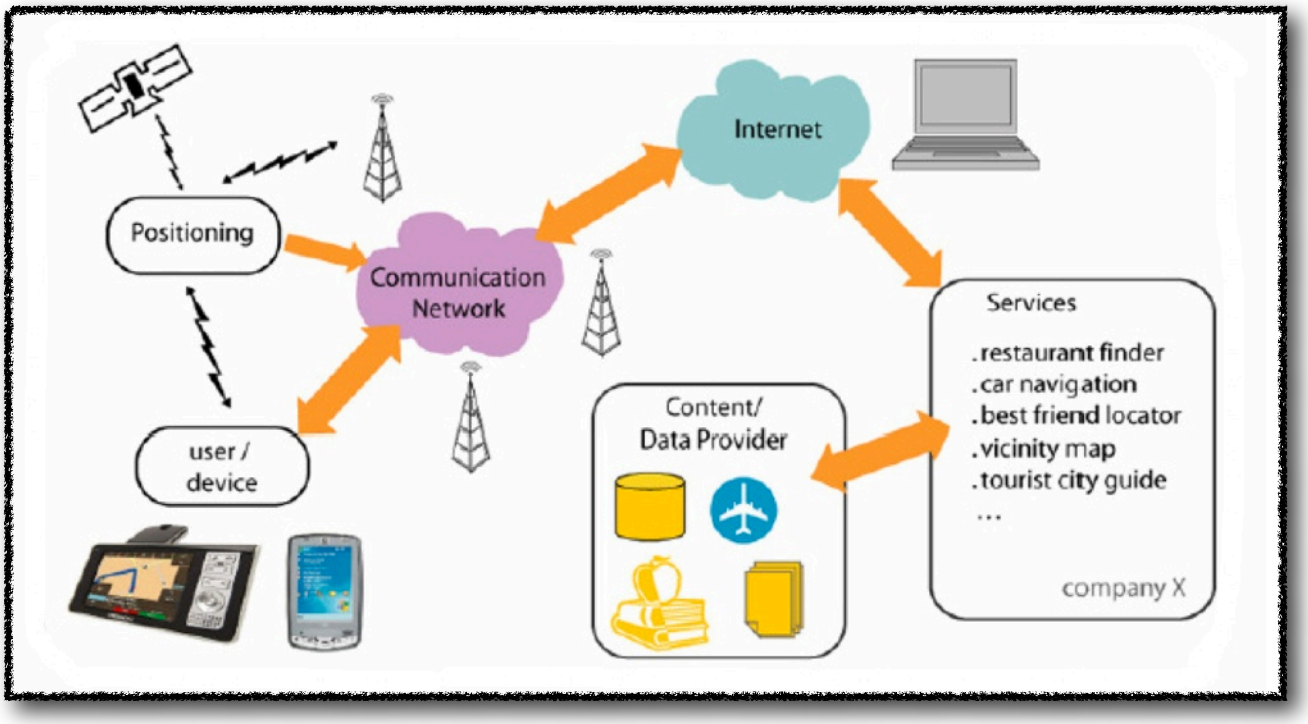

Figura 5: Funcionamiento de LBS[5]

1. El usuario, mediante su dispositivo móvil, pulsa la opción necesaria para obtener los lugares cercanos correspondientes.

2. El dispositivo es geolocalizado (mediante una de las técnicas que se explicarán posteriormente). Tras ello, el terminal envía la petición, que contiene la información que se quiere obtener así como la posición calculada.

3. La puerta de enlace (*gateway*) es la encargada de intercambiar mensajes entre la red de comunicaciones móviles e Internet. Dispone de las direcciones web de diferentes servidores de aplicaciones y envía la petición al servidor correspondiente. La puerta de enlace almacenará así mismo cierta información sobre el dispositivo que ha realizado la petición.

[5] Imagen obtenida de [IV]

4. El servidor de aplicaciones lee la petición y activa el servicio correspondiente, en este caso un servicio de búsqueda espacial.

5. En este punto, el servicio analiza de nuevo la petición y decide qué información adicional necesita tal petición. En el caso expuesto, el servidor pedirá al proveedor de contenidos información adicional sobre lugares turísticos cercanos en la zona.

6. Una vez se dispone de toda la información, el servicio hará una petición para obtener los lugares cercanos. Una vez terminado el cálculo y obtenida la lista de lugares, ésta se envía de vuelta al usuario a través de Internet, la puerta de enlace y la red de comunicaciones móviles.

2.2.4 Aplicaciones de LBS

Los usos que tienen las aplicaciones/servicios de localización son numerosos, aunque se puede ver los cinco ámbitos más comunes en la Figura 6.

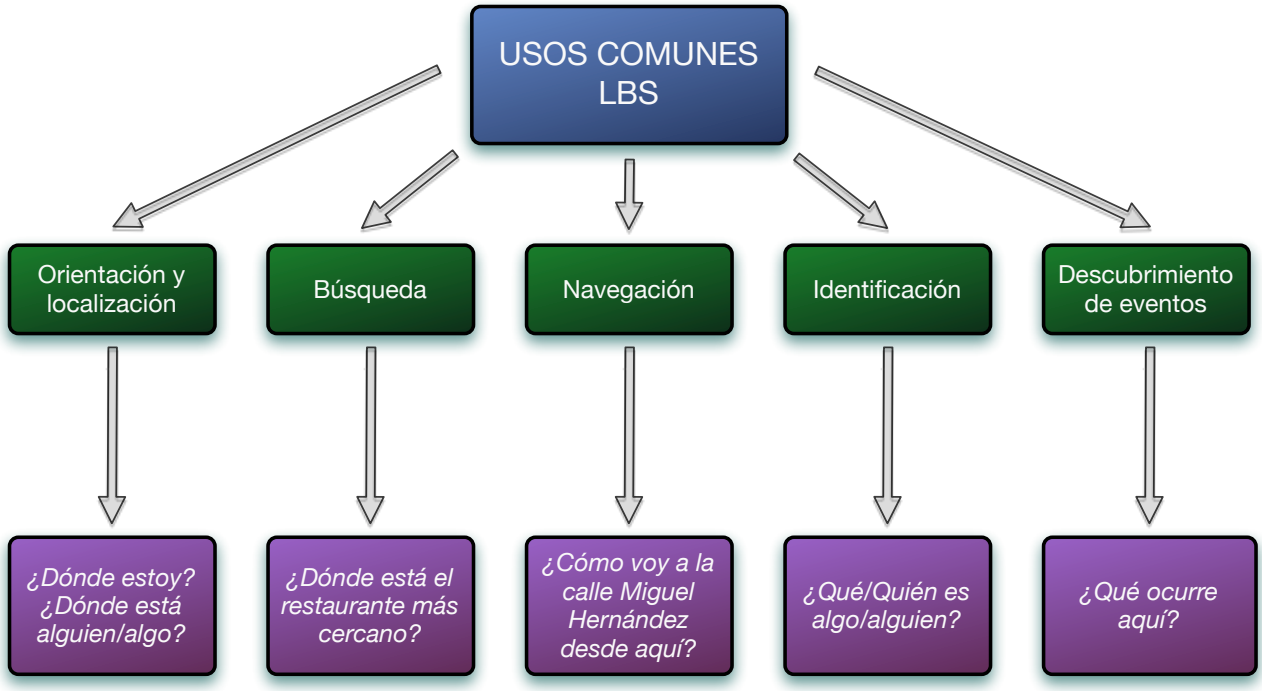

Figura 6: Usos de LBS

El presente proyecto encaja perfectamente en los cuatro primeros grupos:

1. **Orientación y localización**: Trivial. La aplicación ubica al usuario nada más arrancar.
2. **Búsqueda**: Objetivo principal del proyecto. Encontrar lugares turísticos cercanos.
3. **Navegación**: Indicaciones relativas a las rutas entre distintos lugares.
4. **Identificación**: A través de las imágenes/opiniones se puede descubre qué es el lugar.

2.2.5 Técnicas de localización

En esta sección se tratarán tres métodos disponibles para establecer la localización de un terminal móvil. No se adentrará en profundidad en las diferentes técnicas existentes, siendo el objetivo ofrecer una simple panorámica de las mismas.

2.2.5.1 Geolocalización mediante GPS (*General Positioning System*)

La localización mediante esta técnica necesita obligatoriamente tres satélites para realizar la triangulación que determina la posición del terminal (aunque para un posicionamiento mucho más preciso serían necesarios cuatro de ellos).

La precisión que se suele conseguir mediante un dispositivo con GPS suele ser de 1 a 5 metros (la más alta de las tres técnicas que se estudian). Cabe destacar que debido a que el GPS incluido en la mayor parte de los SmartPhones es asistido (A-GPS), la facilidad de encontrar satélites (y por tanto la velocidad de posicionamiento) se mejora muchísimo al utilizar las redes de telefonía móvil para encontrar los satélites cercanos con mayor rapidez que el GPS convencional.

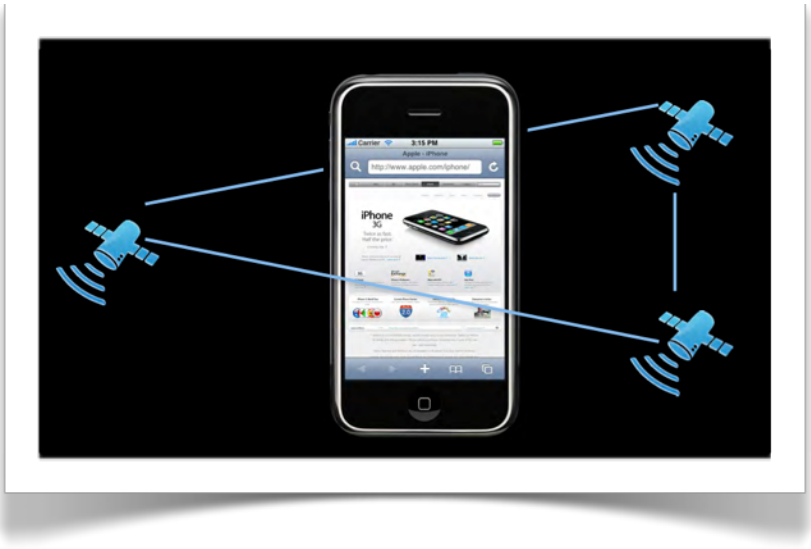

Figura 7: Localización mediante GPS[6]

[6] Imagen obtenida de [V]

2.2.5.2 Geolocalización mediante WPS (*WiFi Positioning System*)

El problema del posicionamiento mediante GPS radica en que, a pesar de ser el método más preciso, la señal empeora significativamente en interiores y en aquellas zonas urbanas con muchos y grandes edificios (lo que aumenta el multitrayecto y por tanto el desvanecimiento de la señal - *fading*). Es aquí donde surge el posicionamiento mediante puntos de acceso wifi (*hotspots*). Consiste en un software que, mediante la dirección IP y la MAC de la conexión WIFI del terminal, obtiene una posición aproximada calculada a través de unas bases de datos que contienen la posición exacta de cada uno de los puntos de acceso wifi existentes. Por tanto, en zonas con numerosos puntos de acceso, el terminal puede hacer la triangulación satisfactoriamente de la misma manera que ocurre con la tecnología GPS. La clara limitación es que dicha base de datos debe ser actualizada y mantenida constantemente para reflejar los cambios producidos en los distintos puntos de acceso.

Aunque no se obtiene tan buena precisión como con tecnología GPS (aproximadamente 20 m), tiene la ventaja de no necesitar visión directa como en el caso anterior, además de ser un método mucho más económico. Destacar la nula aplicación en zonas rurales donde la ausencia de puntos de acceso imposibilita el despliegue de dicha tecnología.

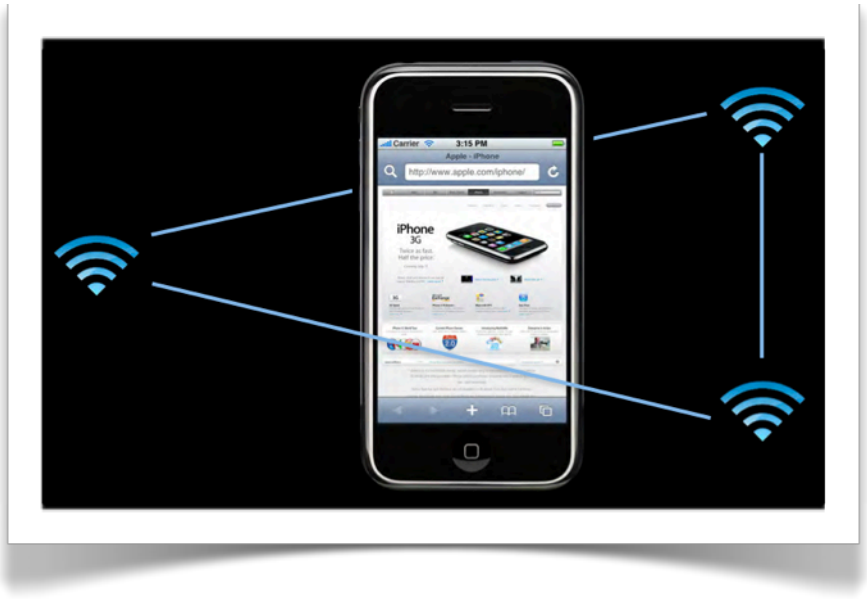

Figura 8: Localización mediante WPS[7]

2.2.5.3 Geolocalización mediante torres de telefonía (*Cell ID*)

Una alternativa a los dos métodos anteriores, aunque menos precisa, es la triangulación mediante torres de telefonía (también conocida como técnica *Cell ID*). Al igual que en el caso de GPS, se necesitan al menos tres para obtener una precisión aceptable. Cada

[7] Imagen obtenida de [V]

torre de telefonía tiene un identificador propio dentro de la red GSM/UMTS y envía unos datos de posicionamiento al terminal. Mediante dichos datos de las torres cercanas y diferentes algoritmos (basados en su mayoría en OTDOA[8] - *Observed Time Difference of Arrival*), se puede llegar a identificar al terminal que se encuentra en el medio de las tres torres utilizadas en el análisis.

La clara ventaja de este método es que no requiere de dispositivos con chips GPS integrados ni de conexión de datos del terminal y por tanto el consumo de batería es notablemente inferior. También existe una gran ventaja y es que posee una cobertura en interiores (*indoor*) mucho mejor que GPS. Aunque la precisión y resolución de este método recae en la densidad de torres de telefonía en la zona, en general, suele ser considerablemente peor que el basado en GPS o WPS (aproximadamente 200 m).

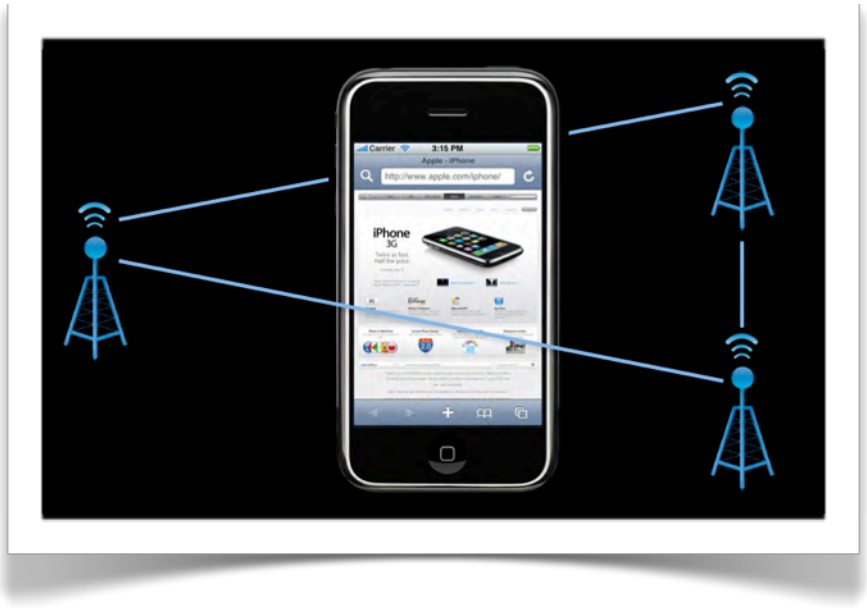

Figura 9: Localización mediante torres de telefonía[9]

En cuanto al método que más se ajusta al uso de la aplicación desarrollada, sin duda se trata de la localización mediante GPS. Esto se debe a que el usuario usará la aplicación en exteriores con el objetivo de buscar lugares cercanos a su posición que, en el ámbito turístico, se tratará de una posición exterior. Aún así, siempre que el turismo se realice en zonas urbanas, con numerosos puntos de acceso wifi, la opción de WPS puede resultar interesante. *Cell ID* ofrece una precisión demasiado baja para ser útil en este caso.

[8] OTDOA se trata de una técnica utilizada en geolocalización de terminales móviles que se basa en la medida de la diferencia de tiempo en la llegada de señales a mismo receptor. Para más información ver http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1635671

[9] Imagen obtenida de [V]

Para terminar esta sección, se mostrarán dos figuras. La primera de ellas es una gráfica comparando la precisión (eje Y) frente a la cobertura (eje X) de varias técnicas de localización. E-OTD (*Enhanced Observed Time Difference*) y TOA (*Time Of Arrival*) quedan fuera del contenido a cubrir por dicho proyecto y pueden obviarse de la Figura 10.

Se puede ver como A-GPS es la técnica que mayor precisión obtiene, teniendo una cobertura en interiores aceptable (mejor que GPS y no tan buena como *Cell ID*). GPS obtiene una precisión buena, pero siempre en exteriores, siendo muy pobre en interiores y zonas urbanas concretas como se explicó anteriormente. *Cell ID* por el contrario, tiene una precisión muy baja, aunque la gran cobertura que ofrece en interiores la convierte en una técnica a tener en cuenta en diversos ámbitos y aplicaciones.

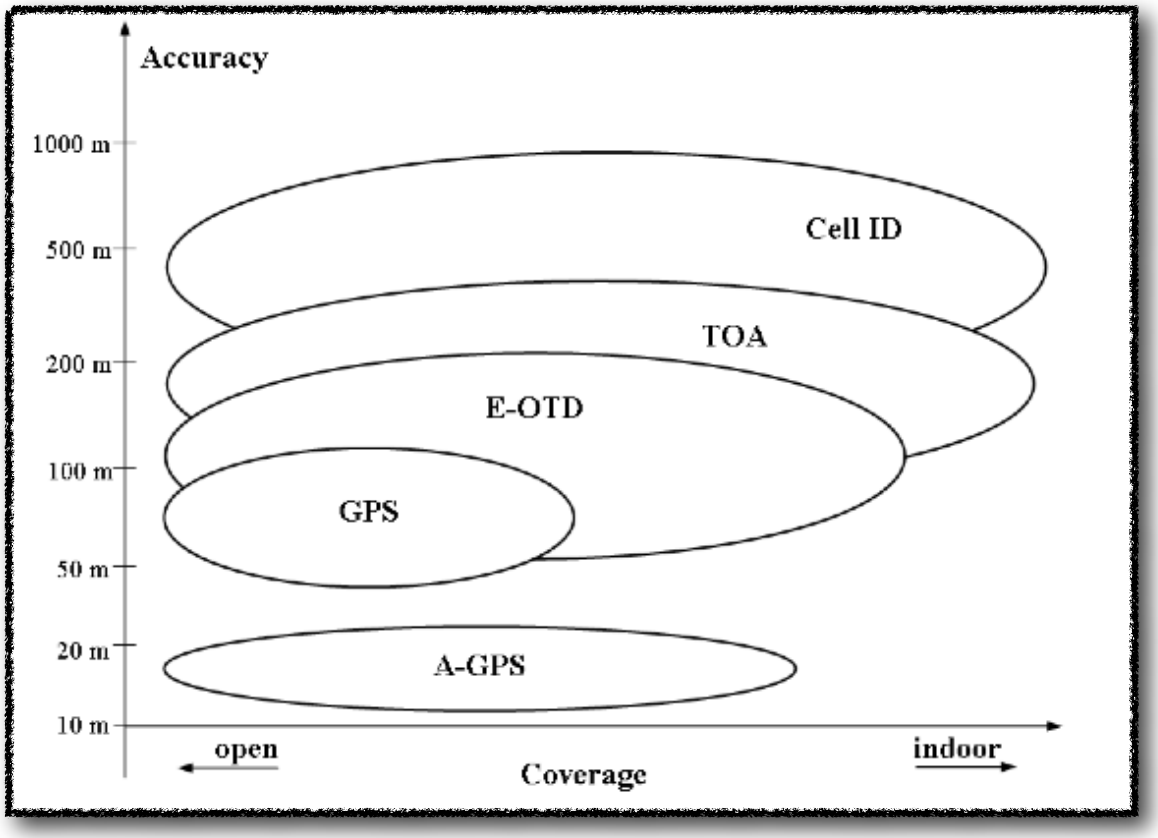

Figura 10: Comparación de métodos LBS[10]

Por último, la Figura 11 muestra el aumento exponencial que han experimentado las aplicaciones basadas en localización. Destacar que en 2010 este crecimiento se ha multiplicado debido a la entrada en el mercado de las aplicaciones LBS más usadas hoy en día, esto es, las aplicaciones de "*check in"* como Foursquare o Gowalla.

[10] Imagen obtenida de [VI]

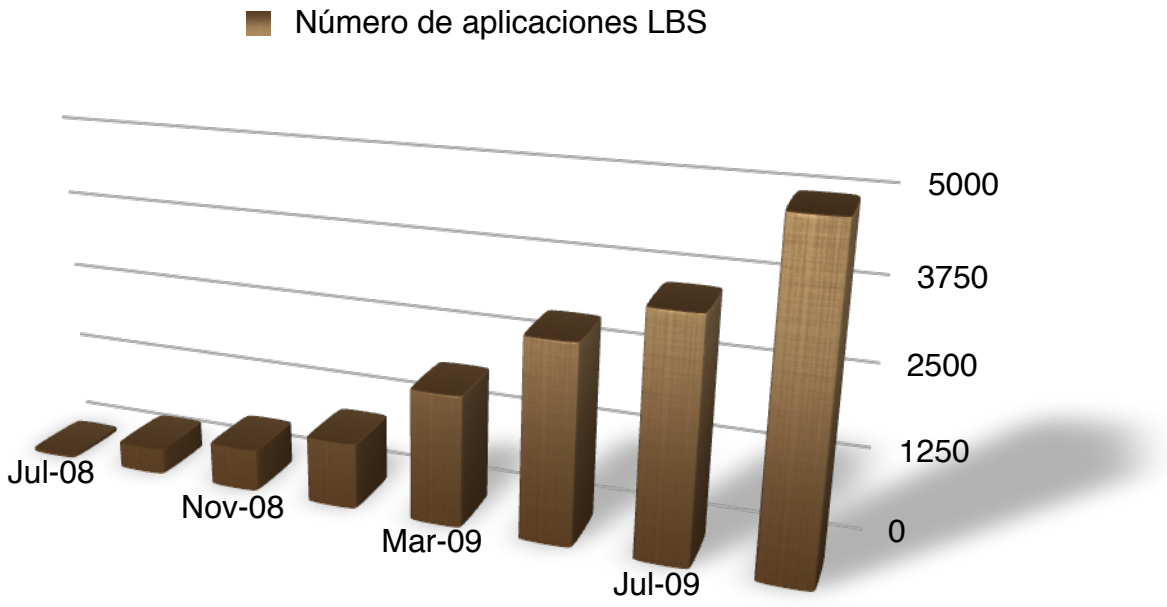

Figura 11: Aumento de aplicaciones LBS[11]

2.3 Comparación de SmartPhones

En esta sección se realizará un análisis del estado actual de los distintos teléfonos inteligentes para llegar a la plataforma final elegida para la realización del proyecto.

2.3.1 Análisis económico

Como se explicará en el capítulo 5, se ha decidido realizar la venta de la aplicación a través de métodos concretos de distribución digital con el fin de obtener beneficios para el propio desarrollador. Se ha descartado la opción de venta a un cliente externo y por tanto se analizará principalmente la necesidad de la utilización de un terminal u otro dependiendo de cual pueda generar mayor ganancia.

El primer requisito importante para el dispositivo en el que funcione la aplicación es que disponga de conexión a Internet (preferiblemente mediante 3G). Muchos dispositivos móviles actuales cuentan con conexión WiFi y aunque el número de puntos de acceso está aumentando (como se comentó previamente), la aplicación está pensada para disponer de una conexión de datos constante y disponible en todo momento. En este punto existen varias posibilidades:

- Conexión GPRS y EDGE (2.5G y hasta 384 Kbps)
- Conexión UMTS (3G y hasta 3.6 Mbps)
- Conexión HSDPA (3.5G y hasta 7.2 Mbps)

[11] Datos obtenidos de [3]

Las conexiones de datos 2.5G son aceptables para tareas que requieran de muy poca descarga de datos (como leer el correo por ejemplo), pero resultan inoperables para un uso normal de Internet. La aplicación desarrollada realiza muchas peticiones a diferentes servicios y páginas Web para obtener la información y por tanto se requiere una velocidad de transferencia aceptable (3G o HSDPA) para conseguir una experiencia de usuario satisfactoria.

Las marcas de terminales más factibles por tanto para realizar la aplicación son:

- Apple (iPhone)
- Google (Android)
- RIM (Blackberry)
- Nokia
- Palm
- Windows

Siempre con el objetivo principal de obtener el máximo beneficio de la aplicación, se compararán a continuación mediante una serie de gráficas las distintas tiendas de aplicaciones de las anteriores compañías.

A través de los datos sacados a la luz por [4] en el Mobile World Congress (MWC) de enero de 2010 en Barcelona, se obtiene la siguiente gráfica:

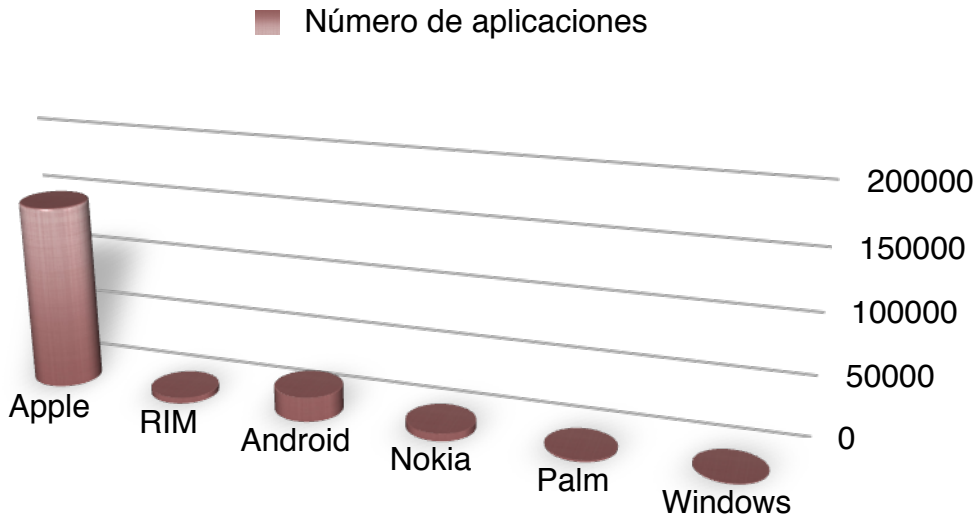

Figura 12: Número de aplicaciones según compañías[12]

Como se puede observar, el número de aplicaciones vendidas por Apple (150.000 aproximadamente) es más de siete veces las de su competidor más directo, Android (20.000 aproximadamente). Mencionar que en el momento en que se están escribiendo

[12] Datos obtenidos de [4]

dichas líneas, el número de aplicaciones de la plataforma de Apple se ha incrementado hasta las 300.000 (generando más de 1 billón dólares en ventas) y Android ha superado las 100.000 (generando algo menos de 100 millones de dólares de ventas). Esto denota un incremento muy importante de Android (aunque lejos de iPhone todavía). Las demás compañías se encuentran todavía muy lejos de esas cifras. Esto da una idea de hacia dónde va el mercado y los desarrolladores. Más aplicaciones implica más desarrolladores y este hecho denota intrínsecamente que la plataforma genera más dinero que las demás.

Esto sería razón suficiente como para elegir la plataforma de Apple frente a las demás. En este punto las dos opciones más interesantes por tanto son Apple y Android, pero antes de elegir la plataforma final de desarrollo del proyecto existen dos factores muy importantes a tener en cuenta como son:

- El porcentaje de aplicaciones gratis y de pago que existen en la plataforma. No tendría sentido adentrarse a conseguir dinero en una plataforma donde el 80% de las aplicaciones son de carácter gratuito.

- El carácter adquisitivo de los usuarios de tal plataforma. Es necesario usuarios que compren aplicaciones.

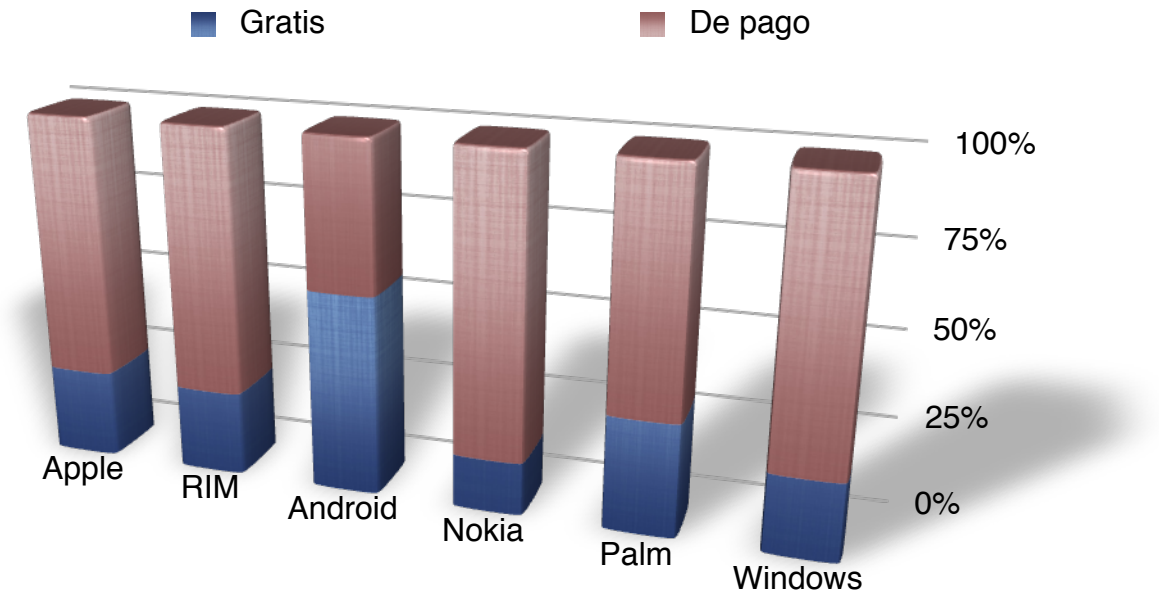

Figura 13: Porcentaje de aplicaciones gratis y de pago[13]

En la Figura 13 se puede ver como la plataforma de Apple es mucho más propensa a tener aplicaciones de pago, siendo este porcentaje un 75% frente al 43% en la plataforma Android. Esto denota de nuevo el alto grado de usuarios existentes en iPhone que realmente compran aplicaciones. De otro modo, los desarrolladores buscarían otros medios de ingresos como la inclusión de anuncios en su software o la utilización del

[13] Datos obtenidos de [4]

sistema iAd del que se hablará en el capítulo 5. En Android este porcentaje es drásticamente inferior.

En las siguientes figuras se puede ver este fenómeno con más claridad.

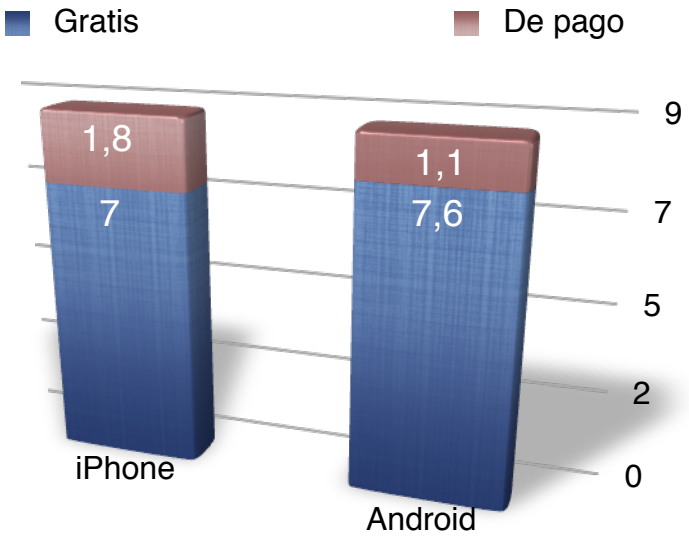

Figura 14: Número de aplicaciones descargadas al mes[14]

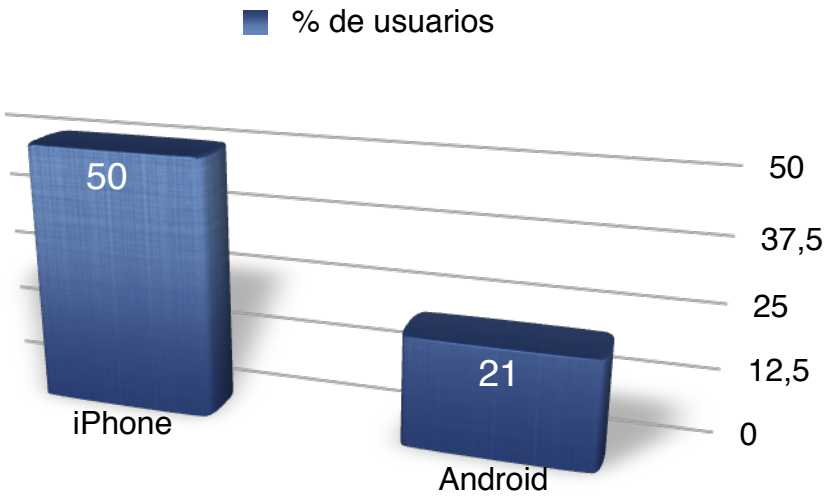

Figura 15: Porcentaje de usuarios que compran una aplicación de pago al mes[15]

Por tanto, se puede comprobar como los usuarios potenciales que realmente compran aplicaciones, y que gastan dinero en complementar su dispositivo con software de terceros, son aquellos que adquieren el dispositivo de Apple.

[14] Datos obtenidos de [4]

[15] Datos obtenidos de [4]

Destacar el problema actual existente en los dispositivos con sistema operativo iPhone, esto es, el *jailbreak.* Consiste en modificar el sistema operativo del dispositivo para que se puedan instalar aplicaciones no permitidas por parte de Apple (bien aquellas rechazadas de la App Store, o aquellas que se encuentran en la tienda de Apple pero de forma ilegal, esto es, sin abonar la cuantía pertinente por la misma). Aún así, el tanto por ciento de dispositivos con *jailbreak* ronda el 10%[16], por lo que, a día de hoy, no es un gran problema ni para Apple ni para los desarrolladores de aplicaciones.

2.3.2 Análisis general entre iPhone y Android

Aún habiendo realizado un análisis económico que posiciona el iPhone como claro candidato para la realización de la aplicación, se hará a continuación un análisis de ciertos aspectos a tener en cuenta a la hora de elegir desarrollar para una u otra plataforma.

2.3.2.1 Desarrollo

Ambas poseen interesantes recursos en Internet para desarrolladores. Android ofrece el SDK (*Software Development Kit*) de forma gratuita [5] y una serie de tutoriales básicos [6]. Apple también permite descargar el SDK de forma gratuita, aunque es necesario registrarse como desarrollador y pagar una cuota anual para acceder a la mayoría de los servicios (como por ejemplo probar la aplicación desarrollada en el dispositivo real) [7].

En cuanto al lenguaje usado, Android usa Java mientras que iPhone está basado en Objective-C haciendo uso del API de Cocoa Touch.

Ambas plataformas poseen herramientas suficientes de test y depuración, así como numerosos foros y una comunidad de desarrolladores suficiente como para obtener la información que se desee a la hora de resolver los diferentes problemas que se tengan.

2.3.2.2 Pago al desarrollador

Para poder subir aplicaciones a la tienda de aplicaciones de iPhone, es necesario pagar una cuota anual de $99. En cuanto a Android, Google requiere un pago de $25 que permite subir aplicaciones a su tienda para siempre. En ambas plataformas, el desarrollador percibe un 70% de las ganancias que genere la aplicación, quedándose el otro 30% restante en Apple/Google.

[16] http://isource.com//2009/11/20/percentage-of-iphone-jailbreak-users-rising-at-almost-10/

2.3.2.3 Visibilidad

El descubrimiento de aplicaciones es uno de los mayores problemas que aparecen en las tiendas de ambas plataformas, donde tal ingente cantidad de aplicaciones tiene el resultado de dificultar que la aplicación desarrollada sea vista por los compradores. Una de las mejores maneras de ser visible es estar el top de las aplicaciones descargadas, lo que en un principio puede resultar difícil y ser necesario publicitar la aplicación en páginas webs, blogs y foros. En un principio, obtener visibilidad en iPhone es más complicado que en Android, debido al mayor número de aplicaciones. Sin embargo, la tienda de aplicaciones de Apple, a la que se accede a través del iPhone o a través de iTunes, suele ser más intuitiva y mejor organizada que la de Android. En cambio, la tienda de aplicaciones de Android es sólo accesible desde el terminal móvil, lo que complica en cierta medida el descubrimiento.

2.3.2.4 Modelo de plataforma

Este es el aspecto que genera más polémica entre las dos plataformas. Apple y su App Store es una plataforma completamente cerrada y controlada por Apple. El desarrollador, tras mandar la aplicación, debe esperar a que sea aprobada por Apple sin ninguna garantía. El 20% de las aplicaciones son rechazadas la primera vez que son mandadas a revisión por el desarrollador, lo que no es nada despreciable. Aún así, Apple indica los aspectos que se deben corregir para que sea aceptada.

En cambio, Android es completamente abierto y acepta prácticamente todas las aplicaciones que no infrinjan algún tipo de copyright.

Esto tiene dos consecuencias:

- El desarrollador puede haber perdido tiempo y dinero desarrollando una aplicación para iPhone que jamás vaya a ser aceptada. Esto en Android no pasa.

- La política estricta de Apple, aunque contraproducente en ciertos aspectos, ha tenido como resultado que las aplicaciones en pasen unos filtros de funcionalidad que, a diferencia de lo que ocurre en la plataforma Android, hacen que el usuario final del dispositivo obtenga una experiencia de usuario mejor en la mayorías de los casos.

2.3.2.5 Fragmentación

La fragmentación es uno de los mayores problemas a los que se enfrentan los desarrolladores de Android. Existen más de cinco versiones de software y más de 60 dispositivos funcionando bajo dicha plataforma. Cada uno de ellos tiene un hardware y pantalla diferente, lo que hace que una aplicación diseñada para los nuevos modelos con Android 2.2 no funcionen en la versión 1.6 (la más extendida). Los desarrolladores tienen que tener en cuenta qué modelos tienen teclados físicos, cuales disponen de cámara y

cuales no... y actualizar sus aplicaciones para cada nueva actualización del sistema operativo.

Por el contrario, en la plataforma iPhone sólo existen cuatro modelos (iPhone 2G/3G/3GS/4) y cuatro versiones del sistema operativo, las cuales son compatibles unas con otras. La mayoría de los modelos tienen características y resoluciones similares (excepto el iPhone 4 que posee el doble, aunque aún así el desarrollo para dicho terminal sigue siendo transparente para el desarrollador) y esta ausencia de fragmentación hace que se obtenga una experiencia mucho mejor de la plataforma debido al enfoque específico que puede implementar el desarrollador.

2.3.3 Elección de la plataforma. Conclusión final.

Se ha podido ver en las dos subsecciones anteriores como las plataformas iPhone y Android son las que actualmente lideran el mercado de terminales móviles. Las principales razones por las que se ha decidido realizar la aplicación para iPhone frente a Android son:

1. El modelo económico. Las aplicaciones para iPhone pueden generar una cantidad de dinero para el desarrollador muy superior a las diseñadas para Android. Al buscarse el mayor beneficio económico posible como se explicó en el capítulo anterior, el iPhone otorga una ventaja que hace que Android se descarte por completo.

2. No se negará la preferencia tanto del que escribe estas líneas, como del tutor del proyecto, por la plataforma Apple. Sin dejar de tomar la decisión de forma objetiva, ha sido un factor importante a la hora de decantarse en la elección de la plataforma, estableciendo una restricción de mutuo entre el tutor y el alumno.

2.4 Aplicaciones similares en la plataforma

Una vez se ha decidido la plataforma de desarrollo de la aplicación, se verá a continuación, y de manera muy rápida, las dos aplicaciones LBS más famosas en la plataforma.

2.4.1 Google Maps

La aplicación de Google Maps para iPhone ofrece unas características muy limitadas con respecto a la versión de escritorio. Entre las principales características, se encuentran:

- Posibilidad de buscar un sitio en concreto.
- Posibilidad de buscar sitios cercanos a la posición del usuario.
- Ver información básica sobre el sitio en cuestión (nombre, dirección, teléfono y web).
- Establecer rutas entre dos lugares.
- Gestión básica sobre los sitios (compartir por correo, añadir a favoritos...).

En la Figura 16 se pueden ver dos capturas de la aplicación de Google Maps para iPhone. En la primera de ellas se observa la posición del usuario mientras que en la segunda el programa marca el punto en el que se encuentra el resultado de la búsqueda (en este caso el Museo del Prado).

Figura 16: Google Maps en iPhone (I)

En la Figura 17 se puede ver el resultado de hacer una búsqueda de museos alrededor de la posición del usuario así como la información que se obtiene del sitio en concreto. Como se puede ver, ésta es muy escasa obteniendo únicamente el nombre y la dirección. También se puede ver como es posible obtener rutas entre dos lugares (y solamente dos).

Figura 17: Google Maps en iPhone (II)

2.4.2 AroundMe

Dicha aplicación es probablemente la aplicación LBS más famosa dentro de la AppStore encontrándose entre las aplicaciones gratuitas más descargadas de la tienda de Apple. Consiste básicamente en una aplicación para buscar lugares cercanos a la posición del usuario (o cualquier otra posición que se elija previamente).

Con respecto a Google Maps, va un paso más allá ofreciendo una lista de categorías de lugares cercanos que pueden resultar de interés, una gestión mejor de los favoritos así como la posibilidad de compartir los lugares encontrados en distintas redes sociales. También permite buscar cualquier lugar que se desee, como el caso de la Figura 18 donde se ha realizado una búsqueda de museos cercanos a la posición actual.

Figura 18: AroundMe en iPhone (I)

En la siguiente imagen se puede ver la información que se obtiene del lugar (nombre, dirección y teléfono/s) así como la posibilidad de mostrar la ruta (abriéndose la aplicación de Google Maps) así como de publicar en las dos redes sociales más famosas.

Figura 19: AroundMe en iPhone (II)

2.4.3 Comparación de la aplicación a desarrollar.

Aunque finalmente se desarrolló la aplicación para que pudiera ser utilizada para la búsqueda de cualquier lugar que desee el usuario, en un primer momento fue muy enfocada hacia el sector turístico. De tal forma se trató de ofrecer un compendio de las mejores características de las aplicaciones mencionadas anteriormente (lo que ya conlleva una dificultad considerable), ofreciendo además características y funcionalidades especiales útiles para cualquier turista que se precie.

Las características diferenciadoras de la aplicación desarrollada sobre las tratadas previamente son:

- Diseño de la aplicación teniendo muy en cuenta la experiencia de usuario en todos los detalles. En dicha plataforma, es algo muy importante y que los usuarios valoran a la hora de comprar una aplicación.

- Obtención de fotos de los lugares desde la propia aplicación.

- Obtención de opiniones de los lugares desde la propia aplicación.

- Obtención de vídeos de los lugares integrados en la propia aplicación.
- Gestión avanzada de favoritos.
- Integración de la tecnología bluetooth para poder enviar los lugares favoritos a otro usuario.
- Gestión de rutas. Posibilidad de obtener la ruta entre más de dos lugares obteniendo indicaciones detalladas para ir de un sitio al otro.
- Posibilidad de guardar la ruta como imagen para uso posterior u offline.
- Posibilidad de ordenar los resultados según distancia o puntuación.
- Diferenciación de los lugares en el mapa según colores basados en la puntuación.

Como se ha mencionado, el proyecto fue diseñado teniendo como objetivo mejorar esas dos aplicaciones tratando de verticalizar la solución hacia el mercado turístico. Para ello se puso el esfuerzo necesario en conseguir las características más importantes de un lugar para el turista (fotos, vídeos, opiniones y rutas). Destacar que, debido a las más de 250.000 aplicaciones que existen en la App Store, es imposible asegurar que la aplicación desarrollada sea completamente novedosa. Por tanto, se han elegido las dos aplicaciones LBS más conocidas, y se ha intentado mejorarlas.

2.5 Análisis de tecnologías: KML, KMZ, XML y JSON

En un primer momento en el inicio de la vida del proyecto, el primer objetivo para obtener los lugares deseados fue estudiar los ficheros generados por herramientas populares como Google Earth, los cuales tienen la extensión KML o KMZ. Se pasará a continuación a explicar en qué consisten este tipo de archivos.

2.5.1 KML y KMZ

El lenguaje KML (*Keyhole Markup Language*) es una modificación de XML (en el cual está basado) para manejar la representación geoespacial de datos en 3D. KML ha sido aceptado como un estándar OGC (*Open Geospatial Consortium*) y es soportado por la mayoría del software que maneja dicho tipo de datos.

En la Figura 20 se puede ver el fichero resultante generado por Google Earth al hacer una búsqueda de museos cercanos a Madrid. Notar que se ha eliminado parte del fichero para su correcta visualización.

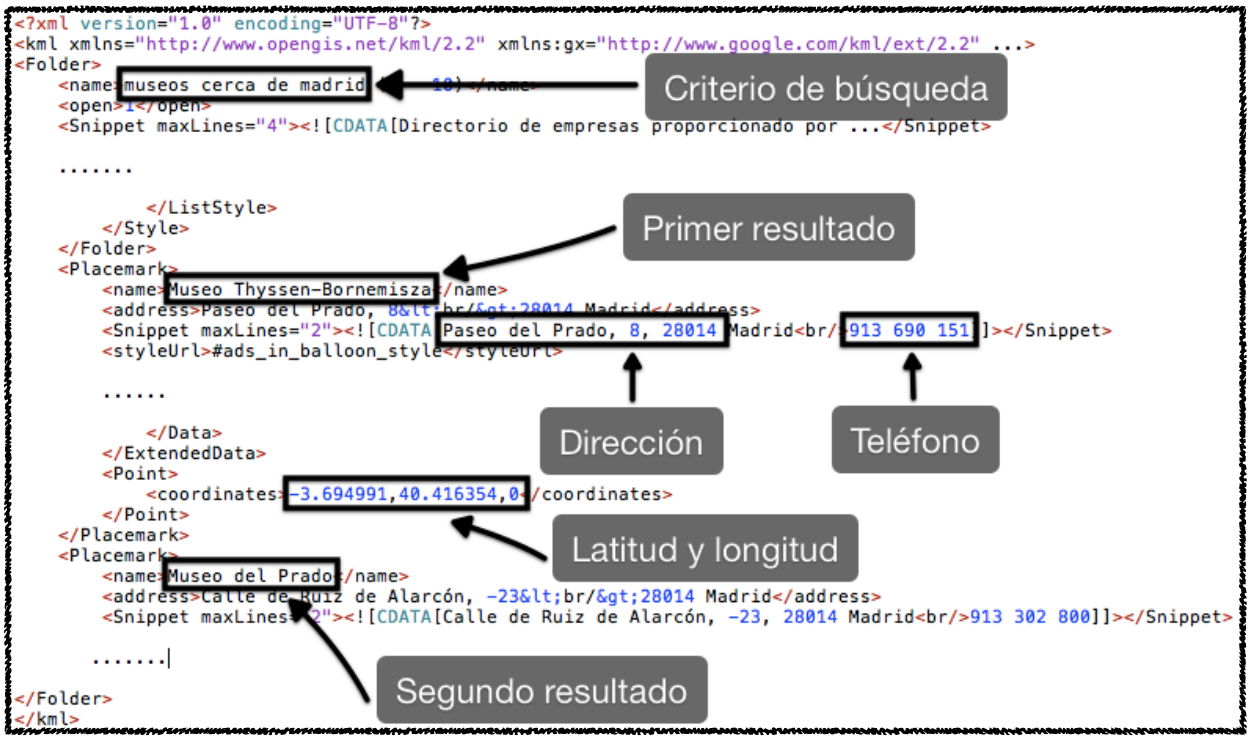

Figura 20: Ejemplo de fichero KML

Como se puede ver, se trata de un fichero XML con ciertas etiquetas asociadas a los lugares (incluida la etiqueta <kml>). Por ejemplo, cada lugar está delimitado por la etiqueta <Placemark>, conteniendo ésta a su vez otras para el nombre del lugar (<name>) o la dirección (<address>).

Los ficheros KMZ consisten simplemente en ficheros KML comprimidos en formato zip.

La utilización de estos ficheros para obtener los resultados necesarios se descartó principalmente debido a las siguientes razones:

- La mayoría de las API de servicios web populares para obtener lugares cercanos a la posición del usuario devuelven los resultados en dos formatos: XML y JSON.

- Debido a lo comentado en el punto anterior, existen numerosas herramientas optimizadas para parsear dicho tipo de ficheros, mientras que el parseo de KML está todavía poco avanzado y en las pruebas realizadas en el proyecto no se consiguió que funcionara de manera satisfactoria.

Para obtener más información sobre KML y KMZ, ver [8], [9] y [10].

2.5.2 XML y JSON

La mayor parte de la información con la que se trabaja en el proyecto proviene de API de servicios web los cuales funcionan según el sistema REST (*Representational State Transfer*). Se trata de una técnica de arquitectura software ligada a sistemas hipermedia distribuidos. También conocidos como sistemas RESTful, basan su funcionamiento en la petición de información por parte del cliente al servidor mediante el protocolo HTTP. En realidad representa un paradigma mucho más complejo gestionando toda la transferencia de dicha información.

Destacar también que no está ligado específicamente al protocolo HTTP, aunque sí es el modo más común de implementación. REST ha tenido un impacto muy importante en la web consiguiendo sustituir a SOAP (*Simple Object Access Protocol*) en la mayoría de las implementaciones debido a su gran sencillez. Si se desea más información, véase [11].

Como se mencionó previamente, las API con las que se ha trabajado (de las que se hablará más adelante) utilizan dos formatos para devolver la información a la petición REST efectuada por parte del cliente, esto es, XML y JSON.

XML (*Extensible Markup Language*) es un lenguaje basado en etiquetas para intercambio de información creado por W3C. Si se desea más información, véase [12] y [13].

Una de las principales desventajas de XML es su complejidad y sobrecarga, lo que hace que parsear este tipo de ficheros consuma muchos ciclos de CPU y sea, en cierta medida, complicado. Aquí es donde surge el formato JSON para solucionar dichos problemas.

JSON (*JavaScrip Object Notation*) es una manera de formatear la información para su intercambio de forma nativa en JavaScript. No está asociado con ningún tipo de patrón de transporte como ocurre con XML. Cuando se recibe la información del servidor, ésta ya está encapsulada en un objeto JavaScript por lo que es mucho más fácil de tratar por parte del cliente.

En la Figura 21 se muestra el fichero JSON obtenido en el navegador tras hacer una llamada al API de búsqueda local de Google, de la que se hablará más detalladamente más adelante.

Figura 21: Ejemplo de fichero JSON

Algunas ventajas de JSON sobre XML son:

- Al ser nativo en JavaScript, simplifica de manera muy importante la evaluación de los datos recibidos. La sentencia *eval()* de JavaScript sobre el String JSON recibido desde el servidor es suficiente para obtener la información buscada.

"*miObjeto = eval('(' + datos_json + ')')*"

- Del anterior punto se deriva que el cliente es mucho más eficiente y rápido.
- Lenguaje más simple, menos etiquetas, menos datos y por tanto más rápido.
- Los mensajes no necesitan ser validados.
- Existen numerosas herramientas para parsear ficheros JSON muy potentes y eficientes.
- Más dinámico, compacto y permisivo que XML.

Si se desea más información sobre JSON, véase [14] y [15].

Debido a todas las razones expuestas arriba y a la existencia de un parseador para JSON muy efectivo en lenguaje Objective-C [16], se ha elegido desarrollar el proyecto utilizando dicho formato.

2.6 Fuentes de datos utilizadas

2.6.1 Obtención de la localización del usuario

En el apartado 2.2 se habló sobre los sistemas basados en localización (LBS) y en el 2.2.6 se explicaron los tres métodos existentes en los teléfonos móviles inteligentes actuales para poder ubicar al usuario. Las herramientas que otorga Apple al desarrollador para llevar a cabo está tarea están basadas en un API o *Framework* llamado *CoreLocation*. Éste tiene un funcionamiento basado en eventos. El teléfono lanza las actualizaciones de posición de forma asíncrona y éstas son recogidas por distintas clases de dicho Framework. Éste es a su vez el encargado de gestionar cualquier tipo de error que se produzca en el cálculo de la posición del terminal. Se tratará más en detalle en el capítulo de implementación. Aún así, al igual que se comentó en el capítulo 1, se destacará el hecho de que la seguridad e integridad del usuario están garantizadas en todo momento. Esto es debido a que el Framework requiere siempre la aprobación explícita del usuario para utilizar su ubicación actual.

2.6.2 API para obtener lugares cercanos

Posiblemente la parte más importante de la aplicación, y en la que se basa todo, es la obtención de los lugares cercanos al usuario. Gran parte del esfuerzo del proyecto se centró en conseguir un API que satisficiera los requisitos buscados.

A continuación se analizarán dichas API y se justificará la elección final.

2.6.2.1 Google Maps

El primer servicio web que se analizó a para conseguir los lugares cercanos a la posición del usuario fue el más conocido, Google Maps. Éste ofrece un API en dos versiones (véase [26]):

1. Versión JavaScript
2. Versión REST

Aunque las aplicaciones iPhone pueden trabajar con funciones JavaScript, resulta mucho más simple y fácil utilizar el API REST y conseguir la información mediante el parseo del fichero JSON resultante. Este será el método de trabajo en dicha API y en las siguientes que se estudiarán.

La petición estándar a utilizar para una búsqueda local es la siguiente:

http://ajax.googleapis.com/ajax/services/search/web/local

En la Tabla 1 se detallan los argumentos más típicos a utilizar en la petición mostrada arriba:

Argumento	Ejemplo	Descripción
q	q=museos+madrid	Especifica la consulta de búsqueda
v	v=1.0	Número de versión
near	near=Madrid ó near=latitud, longitud	Especifica el lugar sobre el cual buscar lugares cercanos
rsz	rsz=large	Especifica el número de resultados (small=4 resultados, large=8 resultados)
start	start=8	Indica el índice del primer resultado de la búsqueda.
hl	hl=es	Indica el idioma en el que se desean los resultados

Tabla 1: Parámetros API Google Maps

Dicha API, a diferencia de las que se verán posteriormente, no requiere de un registro para obtener una llave que dé acceso al servicio. Sin embargo, es útil en el caso de que se necesite soporte por parte de Google.

Supóngase que se desean obtener los museos cercanos a una posición dada. La petición HTTP sería la siguiente:

http://ajax.googleapis.com/ajax/services/search/local?v=1.0&q=museos&near=40.4169186,-3.7034225&rsz=large&start=0&hl=es

El fichero JSON resultante se muestra en la Figura 22.

Figura 22: JSON de lugares Google Maps

Como se puede ver, se obtienen los resultados más comunes que caracterizan a cualquier lugar. Notar que el número máximo de resultados por petición es 8 y por tanto, para obtener los siguientes 8 lugares se debería hacer otra petición, esta vez con el parámetro start=8.

2.6.2.2 Mi nube

El servicio web 2.0 *Minube* [18] trata de convertirse en un repositorio de información turística sobre diferentes ciudades basado en la información proporcionada por los usuarios.

Aún siendo un servicio muy joven, dispone ya de una API [19] con interesantes funciones, las cuáles se analizaron minuciosamente para ver si podían adecuarse a los requisitos del proyecto.

Dicha API, como todas las analizadas en el proyecto, sigue el esquema REST proporcionando un fichero JSON tras la petición HTTP. Como en la mayoría de los casos, es necesario un registro para obtener una llave que da acceso al servicio.

Las distintas opciones que permite esta API son:

- Obtener información sobre un país (nombre, latitud, longitud, código de país, número de hoteles, número de restaurantes...).

- Obtener información sobre una ciudad (misma información que en el caso anterior).

- Obtener la lista de categorías existentes en el servicio.
- Obtener la lista de POI (*Points of Interest*) de un lugar.
- Obtener una lista de comentarios de lugares.
- Obtener lugares cercanos a un lugar o una coordenada dada.
- Obtener fotos sobre un lugar (comentarios inclusive).
- Obtener vídeos sobre un lugar (comentarios inclusive).
- Otras opciones relativas a vuelos (sin interés para el ámbito del proyecto).

Se puede observar como, en un principio, se adecua perfectamente a los requisitos marcados. A continuación se verá el ejemplo típico de utilización que tendría la API dentro del proyecto.

Supóngase que se desea obtener los lugares turísticos cercanos dentro de un radio de 5 Km de longitud en una latitud y longitud marcadas (con un límite de 100 resultados). La petición HTTP sería la siguiente:

http://api.minube.com/places/coordinates.json?api_key=XXXXXXXXXXXXXX
&latitude=40.435156&longitude=-3.692632&category=81&distance=5000&limit=100

Los parámetros a introducir en la petición son:

- Indicar que se desea una lista de lugares cercanos a una coordenada mediante "*places/coordinates*".
- Indicar que el resultado de la petición se desea en formato JSON (por defecto devuelve un XML).
- Latitud y longitud.
- Categoría (en este caso se ha elegido la categoría 81 correspondiente a lugares turísticos).
- Distancia máxima
- Límite de lugares

El JSON obtenido es el siguiente:

Figura 23: JSON de lugares Minube

Como se puede ver, se obtienen las características más importantes del lugar (nombre, coordenadas, dirección, URL, identificador...). Éste último será utilizado para conseguir otros aspectos del lugar como fotos, vídeos u opiniones (se verá más adelante).

Este servicio tiene una gran limitación, y es que no permite buscar un sitio en concreto, sino que simplemente se pueden buscar sitios cercanos basados en categorías. También tiene otra limitación y es que no posee el número de teléfono de casi ningún sitio. A su vez, la base de datos utilizada es bastante pequeña.

2.6.2.3 Qype y 11870

Qype [20] y 11870 [21], son dos importantes servicios web 2.0 de búsqueda de lugares, donde los usuarios pueden dejar sus opiniones, vídeos o fotos sobre los lugares en los que se encuentren. Dichos lugares pueden ser de todo tipo, desde lugares turísticos hasta restaurantes. Las API de dichos servicios se pueden consultar en [22] y [23] respectivamente.

Ambos servicios son más completos que Minube, aunque de cara al proyecto ofrecen características similares, esto es, obtener lugares cercanos a una posición o lugar, junto con las características típicas de los mismos (calle, dirección, latitud, longitud, fotos, vídeos, opiniones...).

Una importante desventaja que tiene el API de 11870 es la ausencia de formato JSON, proporcionando una respuesta XML para todas las peticiones que se haga. Debido a este motivo y a que la API en el momento de su utilización era muy nueva y con diversos fallos, se descartó desde un principio.

En cuanto al API de Qype, ésta ofrece muchas posibilidades entre las que destacan:

- Obtener lugares cercanos a una posición dada.
- Obtener lugares delimitados por un área rectangular.
- Obtener lugares recomendados cercanos a una posición.
- Obtener lugares con amigos.
- Obtener fotos y opiniones de los distintos lugares

Destacar que, como en el caso de Minube, es necesario un registro para obtener una llave necesaria a la hora de hacer las peticiones.

Supóngase que se desea obtener los museos cercanos a una latitud y longitud fijadas. La petición HTTP sería:

```
http://api.qype.com/v1/positions/40.4169186,-3.7034225/places.json?
show=museos&consumer_key=XXXXXXXXXXXXXXX
```

El resultado obtenido es:

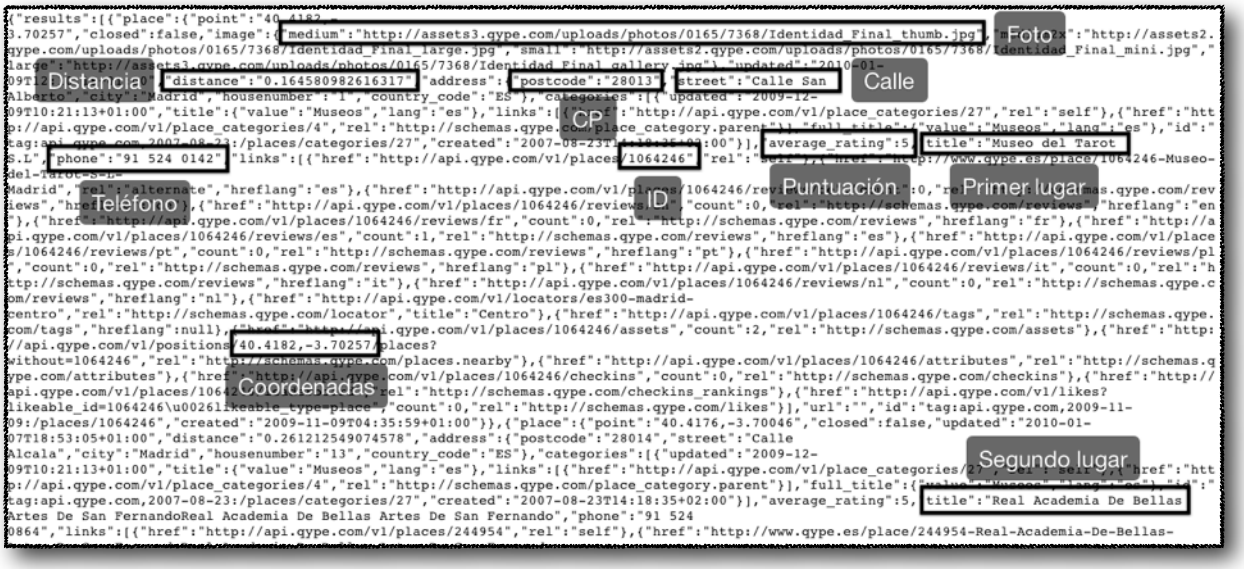

Figura 24: JSON de lugares Qype

Se obtienen prácticamente los mismos resultados que con Minube, aunque ahora las puntuaciones y los números de teléfono parecen estar disponibles en la mayoría de las ocasiones. Aunque lo más importante sin duda, es la posibilidad de obtener las fotos directamente desde el mismo fichero con el que se obtienen los lugares. Esto evita por tanto realizar otra petición y, en consecuencia, acelera la obtención de los datos y la aplicación en general.

2.6.3 API para fotos, vídeos y opiniones

2.6.3.1 Google Maps

2.6.3.1.1 Utilización de las API proporcionadas por Google Maps

La primera opción para conseguir fotos, vídeos y opiniones es la utilización de las API que proporciona Google Maps para tal fin.

API	URL
Fotos	http://ajax.googleapis.com/ajax/services/search/images
Vídeos	http://ajax.googleapis.com/ajax/services/search/video

Tabla 2: API Google Maps para fotos y vídeos

Por ahora no se ofrece un API para conseguir las opiniones de los lugares. Esto descarta en cierto modo esta primera opción al ser un requisito imprescindible. Aún así, se analizará el fichero obtenido cuando se desean obtener fotos de un lugar.

Supóngase que se desean conseguir las imágenes de la Real Academia de Bellas Artes de San Fernando. Debido a que el API devuelve por defecto las mismas imágenes que una búsqueda normal en la web de Google Imágenes, se filtrará para obtener aquellas pertenecientes a Panoramio, las cuales suelen ser las más representativas del lugar. Así, la petición sería:

http://ajax.googleapis.com/ajax/services/search/images?v=1.0&q=real+academia+de+bellas+artes+de+san+fernando&start=0&as_sitesearch=panoramio.com&rsz=large

El JSON correspondiente es:

Figura 25: JSON de fotos en Google Maps

En este caso, el número de imágenes obtenidas es muy escaso, aunque normalmente Panoramio posee una base de datos muy extensa con numerosas imágenes para muchos lugares.

Tras realizar diversos estudios con dicha API, se llegó a la conclusión de que, debido a la imprecisión de la petición al realizarse sólo a través del nombre del lugar, muchas veces se obtuvieron imágenes que no se correspondían con el mismo. Debido a esta razón, y a la ausencia de un API para obtener las opiniones, se planteó una segunda opción que se detalla a continuación.

2.6.3.1.2 Utilización de la información de la web de Google Maps

Cada lugar posee una dirección Web única en Google Maps con las fotos, vídeos y opiniones del sitio. Esta información multimedia es mucho más representativa del sitio que la obtenida con el API anterior. A su vez, las opiniones se componen tanto por aquellas escritas por usuarios de Google Maps, como por las de páginas webs como Qype, Minube, 11870 o TripAdvisor.

En el apartado 2.6.2.1, se obtuvo el fichero JSON con la lista de lugares cercanos junto con un identificador para cada uno. A partir de dicho ID, se puede construir la página web del sitio (tanto para las imágenes, vídeos, como opiniones) según el formato de la Tabla 3:

Recurso	URL
Fotos	**http://maps.google.com/maps/place?cid={ID} &view=feature&mcsrc=photo&num=XX&start=0**
Vídeos	**http://maps.google.com/maps/place?cid={ID} &view=feature&mcsrc=video&num=XX&start=0**
Opiniones	**http://maps.google.com/maps/place?cid={ID} &view=feature&mcsrc=provider_blocks&num=XX&start=0**

Tabla 3: Direcciones Web para obtener fotos, vídeos y opiniones

Estas webs contienen información muy amplia sobre el lugar. Pese a ser un proceso más lento y tedioso que utilizar las API, se decidió parsear el código fuente de dichas páginas para obtener la mejor caracterización posible sobre un lugar.

El aumento del tiempo de espera viene dado por:

1. El analizador JSON utilizado es sustancialmente más rápido que el parseador creado para el código fuente de la página.

2. Para calcular el código fuente, es necesario esperar que la página se cargue por completo.

Se realizaron diversos estudios sobre el impacto que el aumento del tiempo de espera tendría sobre el usuario. Al no ser demasiado grande, se trató de primar la calidad de la información sobre el tiempo en el que el usuario recibe la misma. Aún así, se ha limitado el número de imágenes a 50 (num = 50 en las peticiones) para llegar a un compromiso entre velocidad e información obtenida.

El problema más evidente que surge a la hora de parsear la página Web, es el hecho de que cuando se cambie el formato o diversos parámetros de la misma, el parseador deja de se funcional y debe ser actualizado. En el caso de las API, puede ocurrir también que se renueve y dejen de funcionar (como ha sido el caso de Twitter), aunque es algo menos común. Por tanto, como en el caso de la velocidad, se llegó a un compromiso. Desde que se escribió el parseador alrededor de febrero de 2010, sólo ha tenido que ser modificado una vez para que obtuviera los datos correctamente de nuevo.

Teniendo en cuenta que la aplicación no se ha desarrollado con el único fin de ganar dinero y dejarla olvidada, y no olvidando a su vez que una de las premisas del software 2.0 es el continuo desarrollo del mismo, cambiar el parseador una vez cada cuatro meses parece un precio justo a pagar en favor de que el usuario obtenga la mejor información posible. También decir que cambiar el parseador lleva un coste temporal de no más de media hora, que consiste en identificar los nuevos patrones que delimitan la información multimedia en el código fuente y actualizar el software con éstos.

Por tanto, dicha opción es la elegida. Aún así, se estudian a continuación las distintas posibilidades que ofrecen Minube y Qype para la obtención de recursos multimedia.

2.6.3.2 Minube

Para conseguir fotos y vídeos, la petición REST a hacer es la siguiente:

http://api.minube.com/media/type_of_media.json? api_key=XXXXXXXXXXXXX&poi=XXX

Donde *type_of_media* debe ser reemplazado por la palabra *pictures* o *videos* para obtener las imágenes y vídeos respectivamente del lugar designado por el POI que se indica en la petición.

En la Figura 26 se puede ver el resultado de la petición para obtener las fotos del lugar “Plaza del Dos de Mayo” cuyo identificador de POI es 2693 como se pudo ver en el apartado 2.6.2.2.

Figura 26: JSON de fotos Minube

Para obtener las opiniones, la petición es la siguiente:

http://api.minube.com/places/comments.json?
api_key=XXXXXXXXXXXXX&poi=XXX

El resultado se puede ver en la Figura 27.

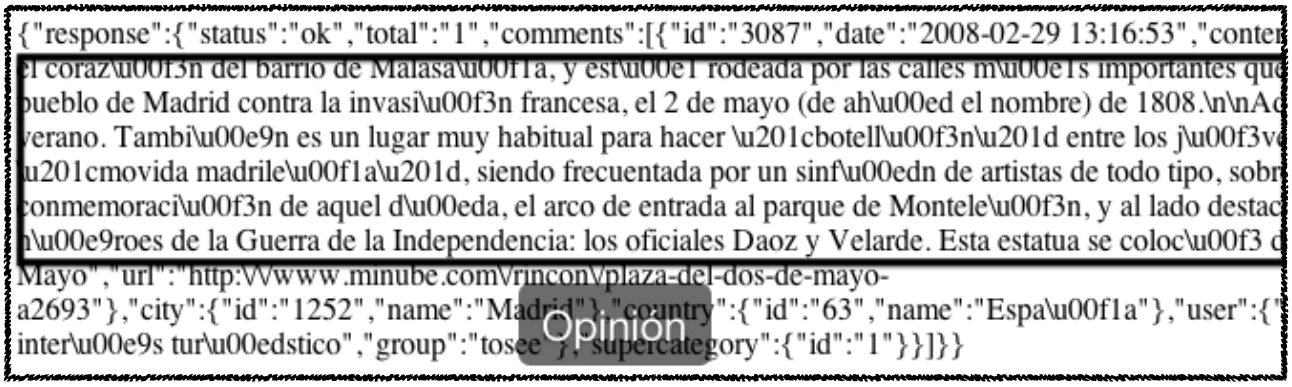

Figura 27: JSON de opiniones Minube

2.6.3.3 Qype

En el caso de Qype, las fotos se obtenían directamente desde el fichero JSON visto en 2.6.2.3. Al no ser posible conseguir vídeos de los distintos lugares a través del API, sólo resta las opiniones para conseguir la caracterización buscada del lugar (los vídeos es la característica menos importante).

Para obtener las opiniones de un lugar con identificador *place_id,* la petición HTTP sería la siguiente:

http://api.qype.com/v1/places/place_id/reviews/es.json?
consumer_key=XXXXXXXXXX

Sustituyendo *place_id* por el identificador visto en el JSON en 2.6.2.3 para el *Museo del Tarot* (1064246), el resultado se puede ver en la Figura 28.

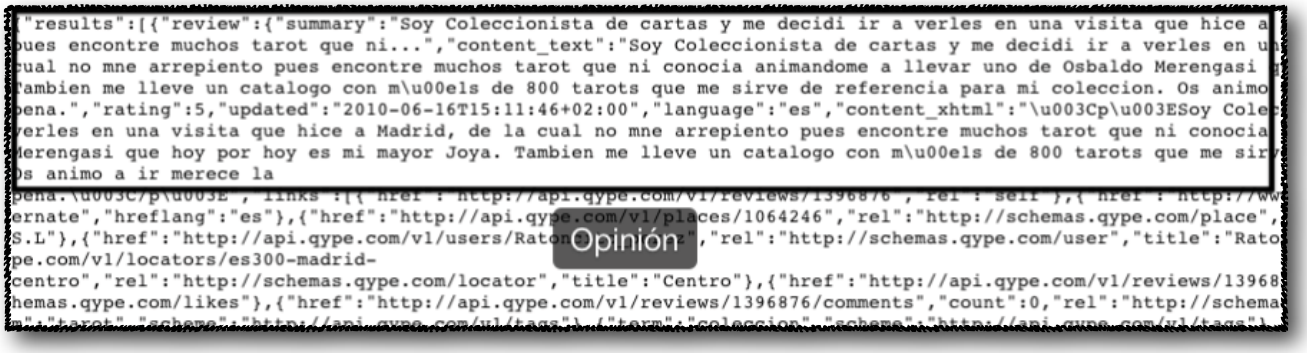

Figura 28: JSON de opiniones Qype

2.6.4 Elección final

A continuación se mencionarán las distintas ventajas y desventajas que poseen las distintas API estudiadas con el objetivo de elegir la más conveniente para el proyecto.

Ventajas y desventajas de Minube y Qype:

- Se adecuan perfectamente al ámbito turístico de la aplicación. En el caso de Minube existe además una categoría específica para lugares turísticos.
- Servicios 2.0 en continuo desarrollo (mejoras sustanciales cada poco tiempo).
- Obtención de toda la caracterización del lugar (características generales, fotos, vídeos y opiniones) directamente desde el API.
- En el caso de Qype, se obtienen las imágenes desde el mismo JSON que los distintos lugares, lo que acelera el acceso a la información de forma considerable al no tener que realizar otra petición para el caso de las imágenes.

Puntos en contra de Minube y Qype:

- Servicios muy nuevos, y por tanto inestables en algunos casos.
- Minube usa un sistema de tickets que otorga 2000 peticiones mensuales. Esto prácticamente descarta dicho servicio.
- Minube no permite buscar un lugar en concreto, sólo búsquedas basadas en categorías.

- Para obtener las imágenes u opiniones, es necesario disponer de un ID único propio del servicio. Esto implica obtener los lugares con el mismo API si no se quieren realizar varias peticiones. Esto es, se tendría que realizar una petición buscando el lugar para obtener el ID, y con éste realizar la petición para obtener las opiniones.

- Base de datos mucho más pequeña que Google Maps.

En relación a Google Maps, destacar que posee una base de datos de lugares mucho más extensa y robusta, obteniéndose la información que proporcionan los dos servicios anteriores (además de estar ordenadas por importancia y popularidad).

Se obtienen igualmente las fotos y opiniones que proporcionan los dos servicios anteriores (además de estar ordenadas por importancia y popularidad), además de ser mucho más precisas. La mayoría de los lugares disponen de fotos de *Panoramio* (servicio web comprado por Google en 2007).

Por contra, la elección tomada de no utilizar el API para fotos y vídeos, hace que el tiempo necesario para conseguir la información (esperar la carga de la página, obtener código fuente de la misma y parseo de ésta) sea notablemente superior al de los anteriores servicios. Aunque perfectamente razonable de cara al usuario.

Aunque los servicios de Qype y Minube ofrecen unas características muy destacables que encajan perfectamente en el ámbito de la aplicación, se deseaba para el proyecto una base de datos extensa y que la información fuera lo más precisa posible. Esto es, se ha primado la calidad de la información sobre la velocidad en la obtención de los datos como se indicó previamente.

Siguiendo el ciclo de vida en cascada con realimentación, en una revisión de la fase de análisis se decidió añadir a la aplicación la funcionalidad adicional de buscar cualquier tipo de lugar genérico que desee el usuario (dejando intacto el enfoque turístico inicial). Esto es, no limitar la aplicación a obtener una lista de lugares turísticos, sino que el usuario pudiera buscar cualquier sitio que desee y se obtengan las mismas características (fotos, vídeos, opiniones, rutas). Esto abre las posibilidades de la aplicación enormemente, pasando de ser una simple aplicación turística, a ser una base de datos de lugares con información extensa sobre los mismos.

Por tanto, tras un lago proceso de análisis y de trabajar con todas las API mencionadas anteriormente, se decidió que la mejor opción para cumplir los objetivos marcados era utilizar los servicios de Google Maps.

2.6.5 Obtención de las rutas

La aplicación de mapas del iPhone ofrece la posibilidad de calcular rutas, sin embargo, posee dos desventajas muy importantes:

- Sólo se puede realizar la ruta entre dos puntos. Al ser uno de los objetivos importantes del proyecto la elaboración de rutas entre diferentes lugares turísticos (más de dos en casi cualquier caso), es algo que descarta esta opción casi por completo.

- Antes de la versión 4.0 del sistema operativo del terminal, éste último no disponía de capacidad de multitarea, por lo que obtener la ruta en la aplicación de mapas implicaba salir de la aplicación en la que se encontraba el usuario (y por tanto cerrarse). Esto es algo muy molesto para el usuario y que se quería evitar.

Se decidió por tanto buscar un servicio del tipo de los analizados anteriormente. Realizar una petición http con dos coordenadas y obtener un JSON con las diferentes latitudes y longitudes que componen la ruta. Esto es justo lo que ofrece CloudMade [24].

El API de CloudMade [25] ha sido utilizada para los siguientes ámbitos:

1 - Obtener la distancia entre dos puntos.
2 - Obtener la ruta entre dos puntos.
3 - Obtener las indicaciones paso a paso entre dos puntos.

Para conseguir dicha información, la petición es la siguiente:

http://routes.cloudmade.com/cloudmade_key/api/0.3/
fromLat,fromLong,toLat,toLong/vehicle.js

Donde (*fromLat, toLat*) componen la coordenada inicial, (*toLat, toLong*) establecen la coordenada final de la ruta y *cloudmade_key* es la llave obtenida al registrarse para usar dicha API.

Vehicle debe ser reemplazado por la palabra *car*, *foot* o *bicycle* para obtener indicaciones en coche, a pie o en bicicleta respectivamente.

Como se puede ver, se indica a su vez el formato de la respuesta justo después del tipo de vehículo (.js = JSON).

A continuación se verá un ejemplo de utilización de la API donde se calculará la ruta para ir desde Atocha (40.40661, -3.68933) hasta la Puerta del Sol (40.41684, -3.70346) en coche. La petición a hacer al servicio de *CloudMade* sería:

http://routes.cloudmade.com/XXXXXXXXXXXXXX/api/
0.3/40.40661,-3.68933,40.41684,-3.70346/car.js

El resultado sería el de la Figura 29.

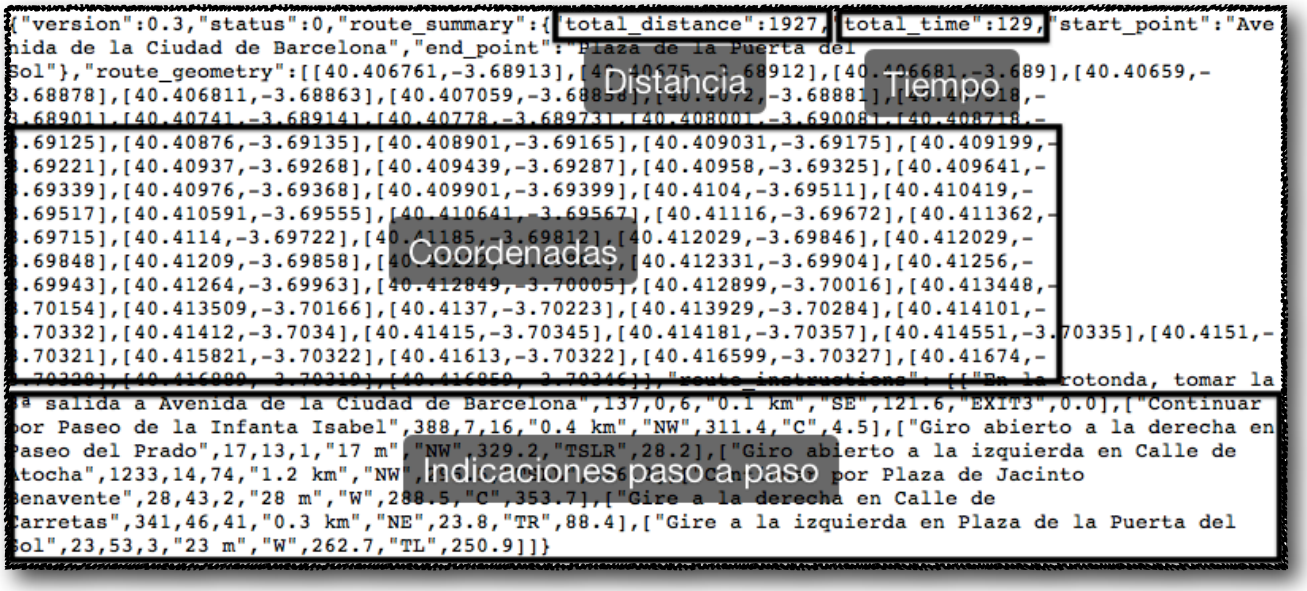

Figura 29: JSON de rutas CloudMade

Como se puede ver, en un mismo fichero se obtiene tanto la distancia, el tiempo de duración, las coordenadas que componen la ruta, como las indicaciones paso a paso para la misma. Las coordenadas obtenidas se dibujaran en la aplicación en una capa superior (*overlay*) a la del mapa, uniendo los puntos mediante rectas para obtener una aproximación muy razonable a la ruta buscada.

Destacar también que siempre se calcula la ruta óptima. Es decir, si el usuario desea visitar tres sitios (llámense A, B y C), la aplicación calculará el lugar más cercano a la posición del usuario, marcando ese lugar como primero a visitar en la ruta. Posteriormente se calculará de nuevo el lugar más cercano de entre los dos que quedan y así sucesivamente.

Se analizaron otros servicios brevemente, aunque ninguno de ellos se adecuaba tanto como *CloudMade* a los requisitos del proyecto. Por tanto esta fue la elección para gestionar las rutas en la aplicación.

2.6.6 Redes sociales

En la actual Sociedad de la Información, uno de los elementos que más adeptos ha conseguido en los último años, son las redes sociales. Desde hace un año, muchas de las aplicaciones para *SmartPhone* vienen dotando de esta capacidad al software.

Para la aplicación desarrollada se quería seguir este camino y dotar a la aplicación con un tinte social tan demandado por los usuarios. Para ello, se eligieron las dos redes sociales más famosas en la actualidad, esto es, Facebook y Twitter.

2.6.6.1 Facebook

Facebook es la red social más grande del mundo con más de 500 millones de usuarios (número que consiguió el verano de 2010). Para integrar este servicio en las aplicaciones, Facebook ofrece un Framework específicamente desarrollado para el iPhone, llamado Facebook iOS SDK [27].

Lo pasos seguidos han sido los siguientes:

1. Crear una aplicación de Facebook y asignar un nombre e icono a la misma (los mismos que se tengan en la aplicación para el iPhone). Se puede ver la página de la aplicación desarrollada en la Figura 30. Se obtiene un número de identificación, una clave del API y una clave secreta.

2. Importar el *Framework* e incluir las cabeceras correspondientes según las indicaciones en [27].

3. Introducir en el código los datos obtenidos en el paso 1.

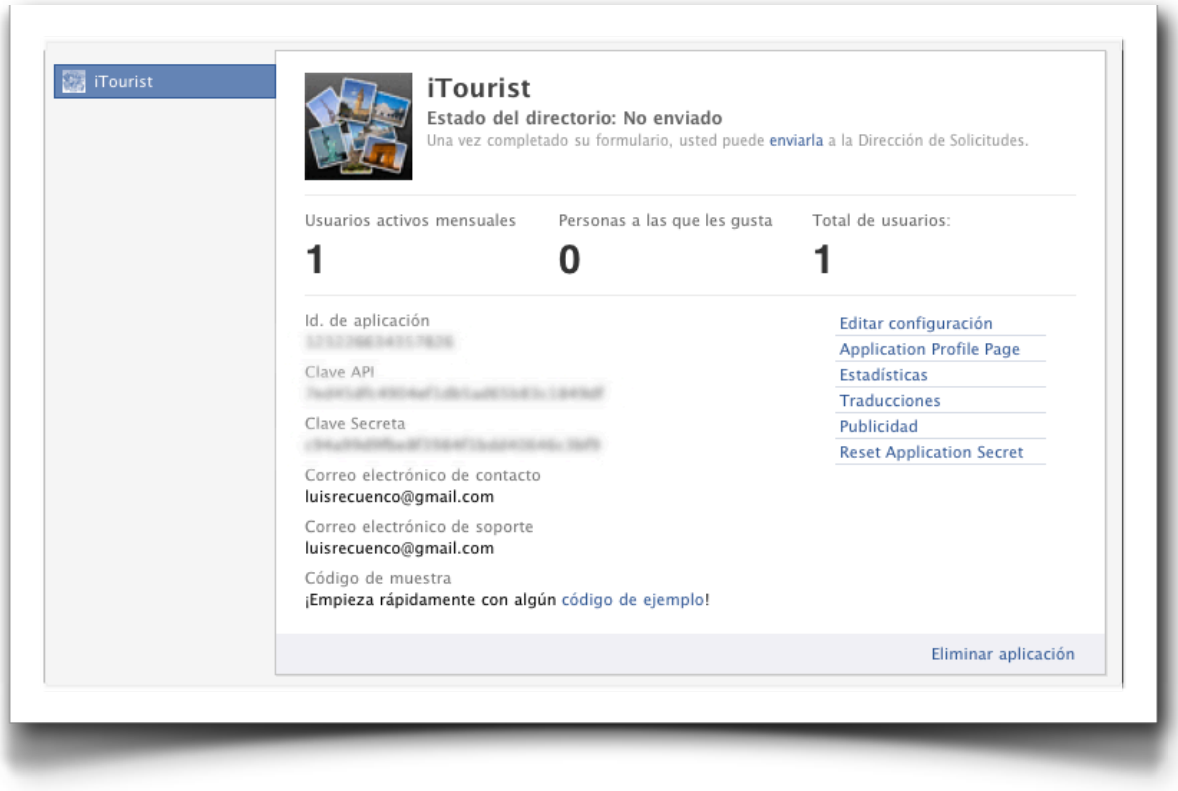

Figura 30: Aplicación iTourist en Facebook

Si se desea ver las pantallas que recibe el usuario a la hora de compartir la información de los distintos lugares en Facebook, véase el Anexo A.

Destacar que el actual *Framework* de Facebook para iOS, a diferencia del utilizado en el proyecto, implementa el protocolo *OAuth*. Éste consiste en un protocolo basado en estándares de software libre que permite la conexión con API de forma segura. La gran ventaja es que permite la autenticación sin que el usuario tenga que dar la contraseña (y por tanto siguen funcionando cuando ésta se cambia). Para más información ver [28].

2.6.6.2 Twitter

Twitter es la segunda red social más importante, con un crecimiento exponencial en los últimos meses. La forma de integrar dicha red social en la aplicación es totalmente diferente a la de Facebook. Twitter no proporciona ningún Framework específico para iOS, por lo que es necesario realizar un código que, a través del nombre de usuario y la contraseña, realice el intercambio de información con el servidor de Twitter. (GET & POST).

En el momento en que se escriben estas líneas, la implementación que se ha realizado ha dejado de funcionar. Esto se debe a que Twitter ya no permite la autenticación básica basada en nombre de usuario y contraseña, habiendo migrado a un modelo más seguro, *OAuth*.

Aún así, en el capítulo de implementación se explicará algo más en profundidad como se ha conseguido la publicación en Twitter mediante dicha autenticación básica.

2.7 Diagrama de casos de uso

En la Figura 31 se detallan los distintos casos de uso que se tienen en la aplicación desarrollada. Como se puede ver, se trata de una funcionalidad muy extensa para tratarse de un dispositivo móvil.

Cada una de las flechas que unen casos de uso denotan el hecho de que ciertos casos de uso pueden englobarse dentro de otros, es decir, el caso de uso del cual parten las distintas flechas da acceso a los casos de uso destino de las mismas. Aunque el usuario sólo tiene interacción con los cinco primeros casos de uso según la Figura 31, en realidad todos y cada uno de los distintos casos de uso se ejecutan con la intervención del usuario, por lo que formalmente se definirían como caso de usos <<*communicates*>>.

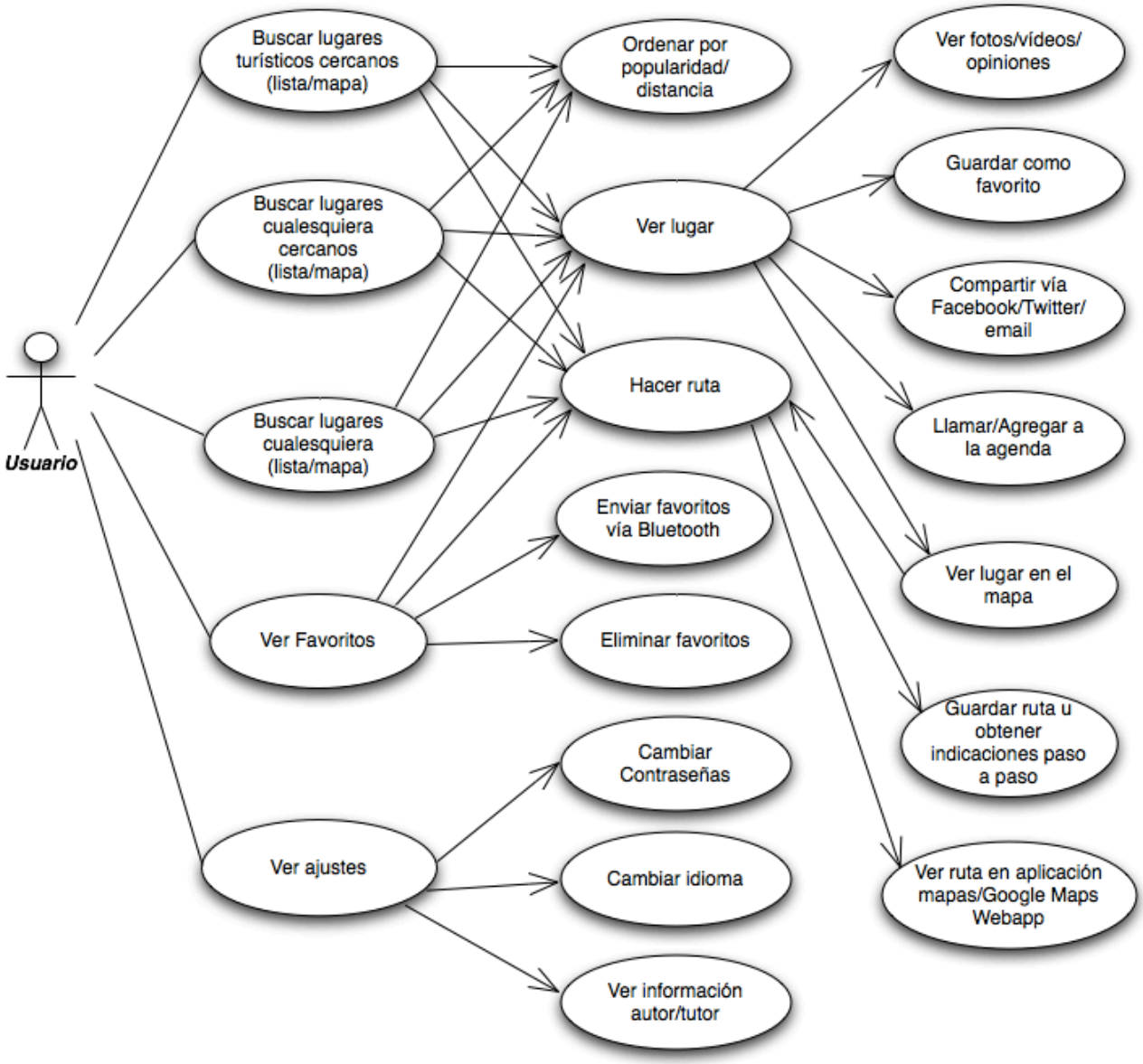

Figura 31: Casos de uso

Como se puede ver, las primeras acciones que el usuario puede realizar con la aplicación están relacionadas con las distintas búsquedas, así como ver los lugares favoritos que se hayan guardado previamente y cambiar los ajustes de la aplicación (contraseña o idioma). Una vez se obtienen los diferentes lugares, las dos acciones más comunes a partir de estos casos de uso son ver un lugar o realizar una ruta. También se pueden ordenar los lugares por popularidad o distancia. Destacar el caso de uso donde se permite un envío de los lugares elegidos por Bluetooth a otro usuario. Otras acciones importantes parten del caso en el que el usuario pulsa sobre el lugar en cuestión, donde se tienen distintas posibilidades como consultar las fotos, vídeos, opiniones, llamar por teléfono o compartirlo en redes sociales.

2.8 Catálogo de requisitos software

En este apartado se detallará la especificación de los requisitos software, tanto funcionales como no funcionales, que ha tenido que cumplir el sistema para concluir el proyecto.

2.8.1 Formato de la especificación de requisitos

Los distintos campos que componen cada uno de los requisitos son:

- **Identificador**: clave que identifica el requisito de manera unívoca.
- **Descripción**: explicación detallada del requisito.
- **Prioridad**: necesidad de la implementación del requisito. Interesante para el desarrollador en su planificación de tareas.
- **Verificabilidad:** identificador de la prueba que verifica el cumplimiento del requisito.

2.8.2 Requisitos funcionales

Identificador	RF-01
Descripción	El programa debe mostrar la ubicación actual del terminal.
Prioridad	Alta
Verificabilidad	PR-01, PR-05

Requisito 1: Ubicación del terminal

Identificador	RF-02
Descripción	El programa inicialmente debe proporcionar al usuario una serie de lugares turísticos cercanos al lugar donde se encuentre.
Prioridad	Alta
Verificabilidad	PR-01

Requisito 2: Obtención de lugares turísticos

Identificador	RF-03
Descripción	El programa permitirá al usuario buscar lugares concretos cercanos a su posición.
Prioridad	Media
Verificabilidad	PR-02

Requisito 3: Obtención de lugares concretos cercanos

Identificador	RF-04
Descripción	El programa permitirá al usuario buscar lugares cualquiera independientemente de su posición.
Prioridad	Media
Verificabilidad	PR-03

Requisito 4: Obtención de lugares concretos

Identificador	RF-05
Descripción	De cada lugar se debe obtener, como mínimo, el nombre, la calle, la latitud, longitud y el número de teléfono.
Prioridad	Alta
Verificabilidad	PR-02, PR-03, PR-08, PR-09

Requisito 5: Obtención de las características básicas del lugar

Identificador	RF-06
Descripción	De cada lugar se obtendrán imágenes que identifiquen al mismo (si existen y hay conectividad a Internet).
Prioridad	Alta
Verificabilidad	PR-03

Requisito 6: Obtención de imágenes del lugar

Identificador	RF-07
Descripción	De cada lugar se obtendrán vídeos que muestren el mismo (si existen y hay conectividad a Internet).
Prioridad	Media
Verificabilidad	PR-03

Requisito 7: Obtención de vídeos del lugar

Identificador	RF-08
Descripción	De cada lugar se obtendrán opiniones sobre los mismos (si existen y hay conectividad a Internet).
Prioridad	Alta
Verificabilidad	PR-04

Requisito 8: Obtención de opiniones del lugar

Identificador	RF-09
Descripción	De cada opinión se tratará de mostrar el nombre, fecha y fuente.
Prioridad	Media
Verificabilidad	PR-04

Requisito 9: Obtención de datos básicos de las opiniones

Identificador	RF-10
Descripción	De cada opinión se tratará de disponer de una URL que lleve al sitio original.
Prioridad	Media
Verificabilidad	PR-04

Requisito 10: Obtención de la página web de la opinión

Identificador	RF-11
Descripción	Cada sitio debe poder ser visto en un mapa.
Prioridad	Alta
Verificabilidad	PR-01, PR-05

Requisito 11: Ubicación de un lugar en el mapa

Identificador	RF-12
Descripción	El programa debe poder proporcionar una ruta desde el lugar en el que se encuentre el usuario hasta el lugar previamente seleccionado por el mismo.
Prioridad	Alta
Verificabilidad	PR-05

Requisito 12: Generación de ruta desde la posición actual

Identificador	RF-13
Descripción	El programa debe permitir establecer rutas entre diferentes lugares genéricos seleccionados por el usuario.
Prioridad	Alta
Verificabilidad	PR-06

Requisito 13: Generación de rutas entre diferentes lugares

Identificador	RF-14
Descripción	El programa debe ser capaz de guardar lugares como favoritos.
Prioridad	Alta
Verificabilidad	PR-06

Requisito 14: Almacenamiento de lugares favoritos

Identificador	RF-15
Descripción	El programa debe ser capaz de compartir los diferentes lugares favoritos por bluetooth con otros usuarios corriendo la aplicación.
Prioridad	Media
Verificabilidad	PR-07

Requisito 15: Envío de favoritos por Bluetooth

Identificador	RF-16
Descripción	El programa debe ser capaz de compartir cualquier lugar en las redes Sociales Facebook y Twitter
Prioridad	Baja
Verificabilidad	PR-09

Requisito 16: Compartir en redes sociales

Identificador	RF-17
Descripción	El programa debe ser capaz de compartir el lugar seleccionado por correo electrónico.
Prioridad	Alta
Verificabilidad	PR-09

Requisito 17: Compartir por correo electrónico

Identificador	RF-18
Descripción	En el caso de que el lugar disponga de número de teléfono, la aplicación debe ser capaz de llamar a dicho lugar.
Prioridad	Alta
Verificabilidad	PR-08

Requisito 18: Llamada telefónica al lugar

Identificador	RF-19
Descripción	Se deben poder ver todos los lugares obtenidos en el mapa al mismo tiempo.
Prioridad	Alta
Verificabilidad	PR-01

Requisito 19: Ubicación de todos los lugares en el mapa

Identificador	RF-20
Descripción	El programa permitirá al usuario ordenar los lugares por popularidad y/o distancia.
Prioridad	Media
Verificabilidad	PR-01

Requisito 20: Ordenar lugares

Identificador	RF-21
Descripción	El programa permitirá agregar un lugar como contacto a la agenda.
Prioridad	Media
Verificabilidad	PR-08

Requisito 21: Agregar lugar como contacto

Identificador	RF-22
Descripción	El programa permitirá guardar la ruta como imagen para consulta offline.
Prioridad	Media
Verificabilidad	PR-05, PR-06

Requisito 22: Guardar ruta

Identificador	RF-23
Descripción	El programa deberá ser capaz de mostrar las indicaciones paso a paso en cada ruta (siempre que lo permita el API utilizada).
Prioridad	Alta
Verificabilidad	PR-05, PR-06

Requisito 23: Indicaciones paso a paso para las rutas

Identificador	RF-24
Descripción	El programa dará la posibilidad de ver la ruta en la aplicación Mapas del iPhone.
Prioridad	Media
Verificabilidad	PR-05

Requisito 24: Obtención de la ruta en la aplicación de Google Maps

Identificador	RF-25
Descripción	El programa avisará al usuario alertando de la ausencia de conexión a Internet.
Prioridad	Alta
Verificabilidad	PR-10

Requisito 25: Conexión a Internet

2.8.3 Requisitos no funcionales

Identificador	RNF-01
Descripción	La aplicación debe ser desarrollada para la plataforma iOS / iPhone.
Prioridad	Alta

Requisito 26: Plataforma de desarrollo

Identificador	RNF-02
Descripción	La aplicación debe ser capaz de de ejecutar los vídeos de los lugares en la propia aplicación, sin tener que depender de la aplicación “Youtube” del iPhone.
Prioridad	Media

Requisito 27: Vídeos dentro de la aplicación

Identificador	RNF-03
Descripción	Se utilizará la base de datos de Google Maps para la obtención de la distinta información sobre los lugares.
Prioridad	Alta

Requisito 28: Base de datos de Google Maps

Identificador	RNF-04
Descripción	Se utilizará, en la media de lo posible, interfaces y controles nativos proporcionados por el SDK del iPhone.
Prioridad	Alta

Requisito 29: Interfaz de usuario

Identificador	RNF-05
Descripción	Se seguirán las líneas de diseño descritas según *iPhone Human Interface Guidelines* [36]
Prioridad	Alta

Requisito 30: Líneas de diseño

Identificador	RNF-06
Descripción	La información de los distintos menús del programa estará traducida al inglés.
Prioridad	Alta
Verificabilidad	

Requisito 31: Traducción a inglés

Identificador	RNF-07
Descripción	La aplicación será compatible con iPhone y iPod Touch con la versión 3.0 del sistema operativo o superior.
Prioridad	Alta
Verificabilidad	

Requisito 32: Compatibilidad de dispositivos

Identificador	RNF-08
Descripción	Todas las pruebas de aceptación deben superarse con éxito.
Prioridad	Alta
Verificabilidad	PR-01 .... PR-10

Requisito 33: Pruebas de aceptación

2.9 Plan de pruebas

En dicho apartado se listará una serie de pruebas que cubren todos y cada uno de los requisitos que se han expuesto en el apartado 2.9. Para considerarse que el proyecto se ha terminado con éxito, todas las pruebas han debido superarse de forma satisfactoria.

2.9.1 Formato del catálogo de pruebas

Los distintos campos que componen cada una de las pruebas son:

- **Identificador**: clave que identifica la prueba de manera unívoca.
- **Descripción**: explicación detallada de la prueba.

Notar que debido la naturaleza de la aplicación, se han omitido campos como la entrada y salida esperada o la precedencia por carecer de sentido en este caso.

También destacar el hecho de que todas las pruebas han sido realizadas con perfecta cobertura GPS. Mencionar a su vez que todas las pruebas (a excepción de la número 10 que comprueba temas de conectividad) se han realizado por duplicado, tanto en cobertura WiFi como en 3G, para comprobar el correcto funcionamiento de la aplicación en ambos casos.

2.9.2 Catálogo de pruebas de aceptación

Identificador	PR-01
Descripción	Se iniciará la aplicación y se comprobará que la lista de lugares obtenida se trata en realidad de lugares turísticos. Se ordenarán por distancia y popularidad. Mediante la segunda pestaña se verán dichos lugares en el mapa y se ubicará al usuario para comprobar que se tratan de lugares cercanos.

Prueba 1: Comprobación de lugares turísticos

Identificador	PR-02
Descripción	Mediante el botón central de la barra superior se buscarán bancos cercanos a la posición calculada justo al iniciar la aplicación. Se pulsará en el lugar deseado y se comprobará que la información básica (nombre, dirección...) es correcta.

Prueba 2: Comprobación de lugares cercanos

Identificador	PR-03
Descripción	En esta prueba se buscarán museos, catedrales y estatuas para posteriormente ver las imágenes que obtiene la aplicación y verificar que se corresponden con el lugar en cuestión. También se verán los vídeos obtenidos y se comprobará que al reproducirse no se ejecuta la aplicación Youtube del iPhone.

Prueba 3: Comprobación de imágenes y vídeos

Identificador	PR-04
Descripción	Se elegirá cualquier lugar de los proporcionados por la aplicación y se pulsará en el botón que lleva a las opiniones del lugar. Se comprueban los datos obtenidos del autor y que la página web (y aquella obtenida a través de *Google Mobilizer*) se corresponden con la opinión obtenida.

Prueba 4: Comprobación de las opiniones

Identificador	PR-05
Descripción	Se elegirá cualquier lugar de los proporcionados por la aplicación y se pulsará en el botón mapa. Se realizará la ruta desde la posición del usuario hasta el lugar correspondiente y se cerciorará el correcto funcionamiento de las indicaciones paso a paso y de la posibilidad de guardar la ruta (saliendo de la aplicación y viendo que la imagen se encuentra en la fototeca del iPhone).

Prueba 5: Comprobación de ruta desde la posición actual

Identificador	PR-06
Descripción	En esta prueba se comprobará la correcta realización de las rutas entre más de dos lugares cualesquiera. Para ello se guardarán tres lugares cualesquiera en favoritos (comprobándose que continúan tras reiniciar la aplicación). Una vez en favoritos se pulsará en el botón para realizar la ruta y se comprobarán todas las opciones disponibles (guardar ruta en fototeca, obtener indicaciones paso a paso...).

Prueba 6: Comprobación de ruta entre varios lugares

Identificador	PR-07
Descripción	En esta prueba se comprobará la correcta transmisión de los distintos lugares favoritos vía Bluetooth. Para ello es necesario disponer de dos dispositivos iOS con la aplicación instalada. Una vez agregados los lugares favoritos deseados, se pulsará en la pestaña *Favoritos* en ambos dispositivos, se comprueba que se está conectado al otro dispositivo y se realiza el envío.

Prueba 7: Comprobación de conectividad bluetooth

Identificador	PR-08
Descripción	Se elegirán tres lugares cualesquiera de entre los proporcionados por la aplicación y se comprueba la correcta realización de la llamada telefónica. También se agrega a la agenda y se comprueba (saliendo de la aplicación) que se ha agregado el lugar como nuevo registro en la aplicación de contactos de iPhone (tanto el teléfono, imagen, nombre y dirección).

Prueba 8: Comprobación de llamada telefónica

Identificador	PR-9
Descripción	Se elegirá un lugar cualquiera de entre los proporcionados por la aplicación y se compartirá tanto en Facebook, Twitter y vía correo electrónico. Se comprobará en Facebook.com y Twitter.com y en el correo correspondiente que se ha efectuado de manera satisfactoria.

Prueba 9: Comprobación de funcionalidad de redes sociales

Identificador	PR-10
Descripción	Estando fuera de cobertura (tanto 3G como WiFi), se iniciará la aplicación y se comprobará que se obtiene el mensaje alertando de la imposibilidad de obtener información.

Prueba 10: Comprobación de conectividad

2.9.3 Matriz de pruebas-requisitos

	PR-01	**PR-02**	**PR-03**	**PR-04**	**PR-05**	**PR-06**	**PR-07**	**PR-08**	**PR-09**	**PR-10**
RF-01	**X**				**X**					
RF-02	**X**									
RF-03		**X**								
RF-04			**X**							
RF-05		**X**	**X**					**X**	**X**	
RF-06			**X**							

	PR-01	PR-02	PR-03	PR-04	PR-05	PR-06	PR-07	PR-08	PR-09	PR-10
RF-07			X							
RF-08				X						
RF-09				X						
RF-10				X						
RF-11	X				X					
RF-12					X					
RF-13						X				
RF-14						X				
RF-15							X			
RF-16									X	
RF-17									X	
RF-18								X		
RF-19	X									
RF-20	X									
RF-21								X		
RF-22					X	X				
RF-23					X	X				
RF-24					X					
RF-25										X

Tabla 4: Matriz de pruebas-requisitos

Capítulo 3: Diseño

En este capítulo se trata de explicar en detalle el diseño de la arquitectura del sistema que se ha llevado a cabo en la realización del proyecto. Así mismo, se presentarán algunos diagramas de secuencia de distintos casos relevantes en la aplicación.

3.1 Diseño del software

3.1.1 Diseño de la arquitectura del sistema

Para la realización del diseño del sistema se ha seguido una arquitectura del tipo "Modelo-Vista-Controlador" (MVC). Esto se debe a que el *Framework* de desarrollo de aplicaciones para iPhone fomenta este tipo de arquitectura que comparten la mayoría de las aplicaciones de la plataforma.

El modelo MVC divide la funcionalidad en tres categorías muy diferenciadas:

- **Modelo**: Las clases que contienen los datos usados por la aplicación.

- **Vista**: Compuesta por las ventanas, controles y otros elementos que el usuario puede ver e interactuar.

- **Controlador**: Junta el modelo y la vista para establecer la lógica que gestiona la interacción del usuario.

La finalidad de este modelo de programación es fomentar la máxima reutilización del código.

En las siguientes figuras, se puede ver fácilmente a qué grupo pertenece cada uno de los componentes. Las clases que pertenecen al grupo "Controlador" siempre llevan al final del nombre de la clase la palabra "*Controller".* Todo lo demás pertenece al modelo, ya que las distintas vistas son atributos propios del mencionado controlador.

A continuación se detalla el diagrama de bloques funcional a alto nivel del sistema. Los bloques principales que se pueden observar son:

1. **Controlador de vistas principales**: Compuesto por los controladores para cada una de las vistas que gestionan las pestañas de la aplicación (ver Anexo A), esto es, el controlador de la vista lista, mapa, favoritos y ajustes. Es el encargado de gestionar la localización del usuario, así como las rutas entre diferentes lugares o cambiar el idioma de la aplicación.

2. **Controlador de la vista Lugar Turístico**: La funcionalidad más importante de la aplicación recae sobre dicho controlador. Es el encargado de gestionar los eventos producidos al pulsar los diferentes botones para mostrar las fotos, vídeos, opiniones, ver el lugar en el mapa o compartir el lugar en las redes sociales.

3. **Lugar turístico:** Módulo principal de la aplicación. Posee interacción con el parseador de lugares (a través del cual se crea) y con el parseador de la información multimedia para conseguir las fotos, vídeos y opiniones del lugar en cuestión.

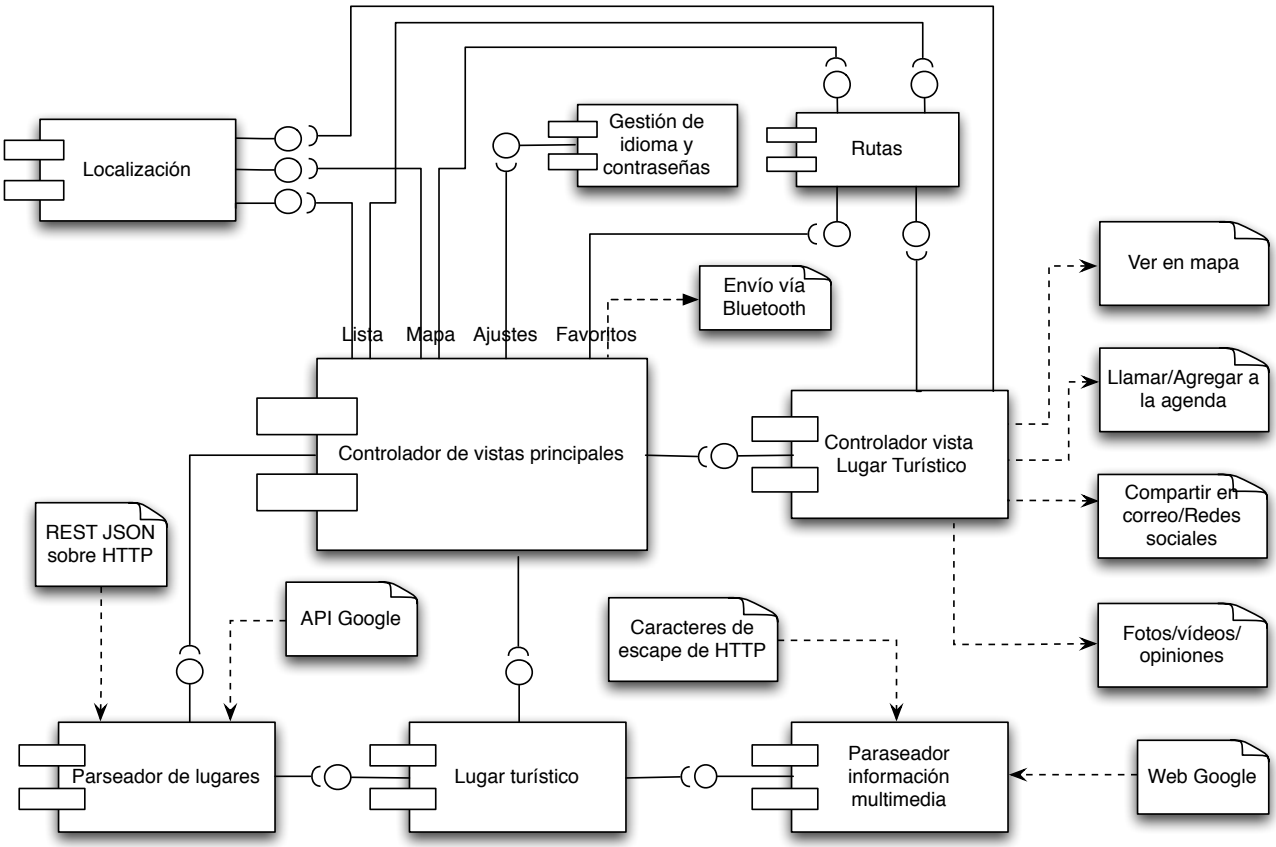

Figura 32: Diseño arquitectónico

En el capítulo 4 de implementación se analizarán en profundidad (a nivel de código) algunas partes importantes de la aplicación como la gestión de las fotos y la funcionalidad bluetooth. A continuación se explicará el diseño de los bloques vistos en la Figura 32.

3.1.2 Diseño del controlador de vistas principales

Una de las partes más importantes de la aplicación fue el diseño del controlador de vistas principales. Es una parte a la que se dedicó un tiempo considerable a través de la realización de un análisis basado en las necesidades de la aplicación, lo que ésta tenía que ofrecer, y la interacción que el usuario haría de la misma. Se trató de buscar siempre la sencillez y simplicidad, siguiendo las guías de diseño descritas por *iPhone Human Interface Guidelines* [36]. La aplicación está diseñada según cuatro partes muy diferenciadas:

1. Ver los diferentes lugares en una lista.
2. Ver los diferentes lugares en un mapa.
3. Gestión de favoritos.
4. Ajustes.

Otro aspecto importante listado en los requisitos no funcionales del capítulo tres fue la utilización de controles nativos de la plataforma. El SDK del iPhone ofrece un controlador de vistas que se adapta perfectamente a los requerimientos de la aplicación y que está basado en pestañas, una para cada uno de los puntos listados arriba. Dicho controlador recibe el nombre de *UITabBarController* y posee los controladores para cada una de las cuatro vistas mencionadas.

Antes de entrar en detalle en el diseño de estos controladores, se analizarán las distintas clases por las que pasa la aplicación hasta cargar los mismos, las cuales se pueden ver en la Figura 33.

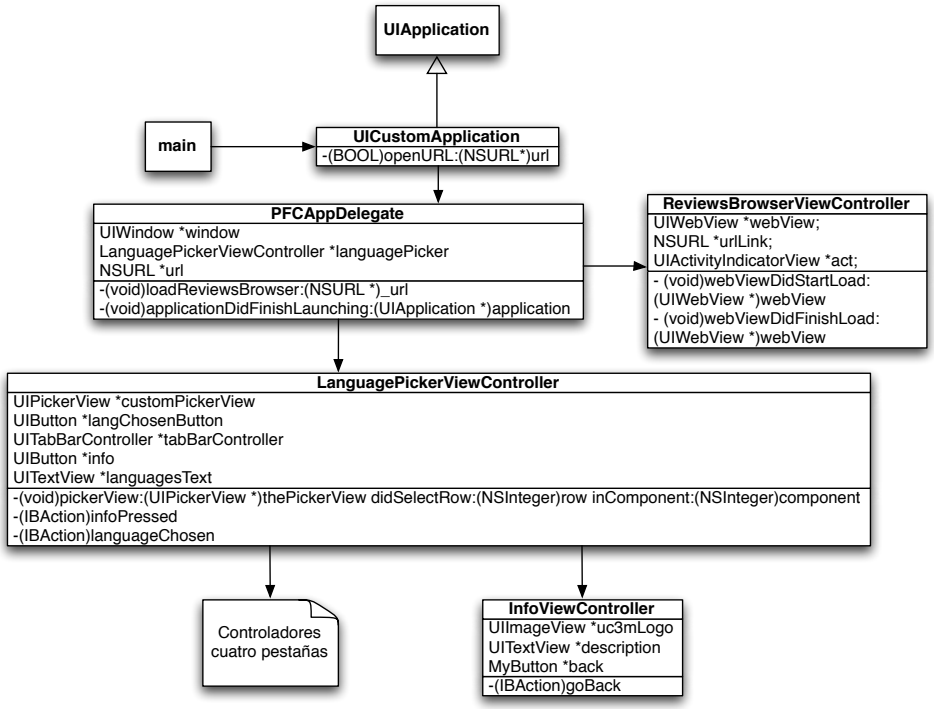

Figura 33: Clases participantes en el inicio de la aplicación

La función principal de las distintas clases es:

- ***main*:** Fichero en el que comienza toda aplicación para iPhone. Contiene un único método al que se indica el nombre de la clase *UIApplication.*

- ***UICustomApplication*:** Permite al sistema abrir los diferentes enlaces a páginas web dentro de una vista de la propia aplicación.

- ***PFCAppDelegate*:** Aunque la aplicación empieza gracias a *UICustomApplication,* ésta delega toda la funcionalidad a la clase *UIApplicationDelegate*, protocolo[17] implementado por la clase *PFCAppDelegate.* Ésta última es la encargada de cargar la primera ventana de la aplicación.

- ***LanguagePickerViewController*:** Encargada de la selección del idioma deseado.

- ***InfoViewController*:** Clase cuya única funcionalidad es mostrar información sobre el proyecto (tutor y alumno).

- ***ReviewsBrowserViewController*:** Encargada de gestionar la vista Web de las diferentes opiniones sobre el lugar.

A continuación se detallan los atributos y métodos de las diferentes clases mencionadas.

PFCAppDelegate	
Atributos	
window	Ventana principal de la aplicación. Todas las vistas se superponen sobre dicha ventana.
languagePicker	Selector de idioma.
url	Página web de la opinión del lugar.
Métodos	
loadReviewsBrowser	Carga la vista con la web de la opinión en la propia aplicación y gestiona el uso de *Google Mobilizer*.
applicationDidFinishLaunching	Carga del controlador del selector de idioma (*LanguagePickerViewController*).

Tabla 5: Clase PFCAppDelegate

[17] Un protocolo en lenguaje Objective-C es el “equivalente” a una interfaz en el lenguaje Java

LanguagePickerViewController	
Atributos	
customPickerView	Vista del selector de idioma.
langChosenButton	Botón usado para la selección de idioma y cargar la vista principal del controlador UI*tabBarController.*
tabBarController	Controlador de las cuatro pestañas de la aplicación.
info	Botón para cargar la vista de información de la aplicación (*InfoViewController*).
languagesText	Descripción sobre la elección de idioma en los cinco lenguajes de la aplicación.
Métodos	
pickerView:didSelectRow:inComponent	Método ejecutado cada vez que se elige un idioma con el selector.
infoPressed	Método llamado al pulsar el botón *info*.
languageChosen	Método llamado al pulsar el botón *langChosenButton*.

Tabla 6: Clase LanguagePickerViewController

infoViewController	
Atributos	
uc3mLogo	Imagen del escudo de la universidad.
description	Descripción sobre el alumno y el tutor del Proyecto en los cinco idiomas de la aplicación.
back	Botón para volver a la vista de selección de idioma.

infoViewController	
Métodos	
goBack	Método llamado al pulsar sobre el botón *back.*

Tabla 7: Clase InfoViewController

ReviewsBrowserViewController	
webView	Vista de la Web de la opinión.
Atributos	
urlLink	URL a cargar por la vista Web.
act	Indicador de carga.
Métodos	
webViewDidStartLoad	Método llamado al comienzo de la carga de la Web.
webViewDidFinishLoad	Método llamado a la finalización de la carga de la Web.

Tabla 8: Clase ReviewsBrowserViewController

Como se puede ver, en la vista de selección de idioma, se tienen dos posibilidades:

1. Seleccionar el idioma deseado para obtener a continuación los lugares turísticos cercanos de la zona (cargar el controlador *UITabBarController*).

2. Seleccionar información sobre la aplicación, donde se puede ver el nombre del alumno y el tutor (cargar el controlador *InfoViewController*).

En la Figura 34 se detalla la estructura de los cuatro controladores que forman la estructura principal de la aplicación. Notar que sólo se mostrarán los atributos y métodos más importantes de las cuatro clases.

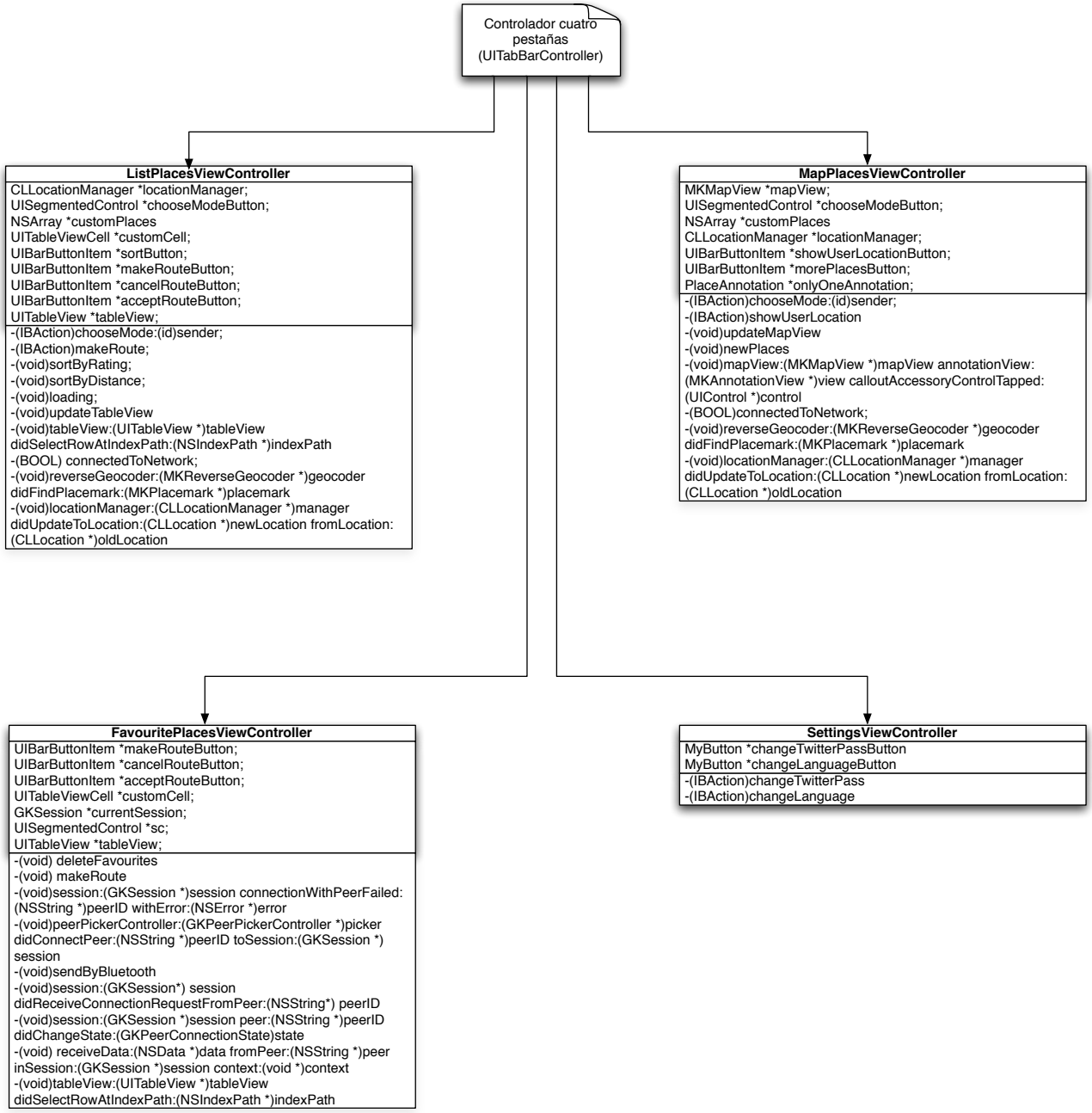

Figura 34: Diseño del controlador de vistas principales

La función principal de las distintas clases es:

- ***ListPlacesViewController*:** Localiza al usuario y permite buscar los diferentes lugares deseados y visualizarlos en modo de lista. Permite ordenarlos según distancia o popularidad, a la vez que realizar rutas entre los mismos.

- ***MapPlacesViewController*:** Localiza al usuario y permite buscar los diferentes lugares deseados y visualizarlos en un mapa.

- ***FavouritePlacesViewController*:** Permite visualizar en modo de lista los diferentes lugares que se han guardado en la aplicación. Permite realización de rutas y envío de los mismos vía Bluetooth.

- ***SettingsViewController*:** Cambio del idioma de la aplicación y la contraseña para la cuenta de Twitter utilizada.

A continuación se especificarán los distintos atributos y métodos existentes en dichas clases, acompañados de una breve descripción de los mismos.

Atributos y métodos comunes (ListPlacesViewController/MapPlacesViewController)	
Atributos	
locationManager	Encargado de gestionar la localización del usuario.
chooseModeButton	Vista de los tres botones encargados de realizar las búsquedas en la aplicación.
customPlaces	Array con los lugares a mostrar en la lista/mapa.
Métodos	
chooseMode	Método llamado al pulsar el botón *chooseModeButton.*
connectedToNetwork	Comprueba que existe conexión a Internet (vía WiFi, 3G o GPRS).
loading	Muestra la animación *QuartzCore* para alertar al usuario de la carga de lugares.
reverseGeocoder:didFindPlacemark	Método llamado cuando la geocodificación inversa se ha realizado con éxito.
locationManager:didUpdateToLocation:fromLocation	Método llamado cuando se actualiza la posición del usuario.

Tabla 9: Métodos y atributos comunes (I)

Atributos y métodos comunes (ListPlacesViewController/FavouritePlacesViewController)	
Atributos	
(make/cancel/accept)RouteButton	Encargados de iniciar/cancelar el cálculo de las rutas.
customCell	Contiene el objeto celda con la disposición específica diseñada (foto del lugar, calle, puntuación...)
tableView	Lista con las celdas de lugares correspondientes al atributo explicado arriba.
Métodos	
makeRoute	Realiza la ruta entre los lugares seleccionados.
tableView:didSelectRowAtIndexPath	Carga el controlador del lugar seleccionado.

Tabla 10: Métodos y atributos comunes (II)

ListPlacesViewController	
Atributos	
sortButton	Botón para ordenar los lugares por distancia o popularidad.
Métodos	
sortByRating	Ordena los lugares por popularidad
sortByDistance	Ordena los lugares por por distancia
updateTableView	Método más importante de la clase. Encargado de llamar al parseador de lugares y conseguir los mismos para poblar la lista con los lugares correspondientes.

Tabla 11: Clase ListPlacesViewController

MapPlacesViewController	
Atributos	
mapView	Vista correspondiente al mapa.
showUserLocationButton	Botón para mostrar la posición del usuario.
morePlacesButton	Botón para obtener los siguientes ocho lugares de la búsqueda.
onlyOneAnnotation	Objeto para discernir la utilización de dicho controlador (se explica más abajo)
Métodos	
showUserLocation	Método llamado al pulsar sobre *showUserLocationButton.*
newPlaces	Método llamado al pulsar sobre *morePlacesButton*.
updateMapView	Método más importante de la clase. Encargado de llamar al parseador de lugares y conseguir los mismos para poblar el mapa con los lugares correspondientes.
mapView: annotationView:calloutAccessory ControlTapped	Carga el controlador del lugar seleccionado.

Tabla 12: Clase MapPlacesViewController

FavouritePlacesViewController	
Atributos	
currentSession	Sesión necesaria para la funcionalidad Bluetooth.
sc	Botón encargado de iniciar la búsqueda de dispositivos bluetooth cercanos corriendo la aplicación.
Métodos	
session:connectionWithPeerFail ed:withError	Método llamado al ocurrir un error en la funcionalidad Bluetooth.

FavouritePlacesViewController	
peerPickerController:didConnectPeer:toSession	Método llamado al conectarse vía Bluetooth.
sendByBluetooth	Encargado de realizar el envío por Bluetooth.
receiveData:fromPeer:inSession:context	Gestiona la recepción de los lugares por Bluetooth.

Tabla 13: Clase FavouritePlacesViewController

SettingsViewController	
Atributos	
changeTwitterPassButton	Botón para cambiar la contraseña de Twitter.
changeLanguageButton	Botón para cambiar el idioma.
Métodos	
changeTwitterPass	Método llamado al pulsar sobre *changeTwitterPassButton.*
changeLanguage	Método llamado al pulsar sobre *changeLanguageButton* (carga el controlador *LanguagePickerViewController* visto en la Figura 33).

Tabla 14: Clase SettingsViewController

Una peculiaridad de la clase *MapPlacesViewController* es el atributo *onlyOneAnnotation.* Como se mencionó en la Tabla 12, éste es usado para discernir la utilización del controlador debido al doble uso que se le da al mismo en el sistema. Dicho controlador es usado para:

1. Funcionalidad mencionada arriba, esto es, ver todos los lugares obtenidos en el mapa al mismo tiempo.

2. Ver un lugar en concreto geoposicionado en el mapa.

En vez de crear una nueva clase que fuera el controlador de la vista que se carga cuando el usuario pulsa el botón mapa en la vista específica del lugar seleccionado, se decidió reutilizar dicha clase ya creada y diferenciar su utilización a través del atributo *onlyOneAnnotation* (que en el primer caso tendrá valor NULL y en el segundo se corresponderá con el lugar concreto que se desea representar). En el diagrama de la

Figura 34, por tanto, sólo se han mostrado los métodos y atributos correspondientes a la primera funcionalidad del controlador.

3.1.3 Diseño del controlador de rutas

Otra parte importante en la aplicación es la gestión de rutas. La aplicación permite dos tipos:

1. Ruta realizada por el controlador *MapPlacesViewController.* Dicho tipo de ruta se verá más adelante. Es accedida cuando se pulsa el botón mapa en la vista específica del lugar y sólo permite realizar la ruta desde la posición del usuario hasta dicho lugar.

2. Ruta realizada por el controlador *MapRouteViewController.* Este tipo de ruta es más general, permitiendo rutas con un número de lugares indeterminado (incluyendo también la posición del usuario).

El diagrama de clases correspondiente se puede ver en la Figura 35.

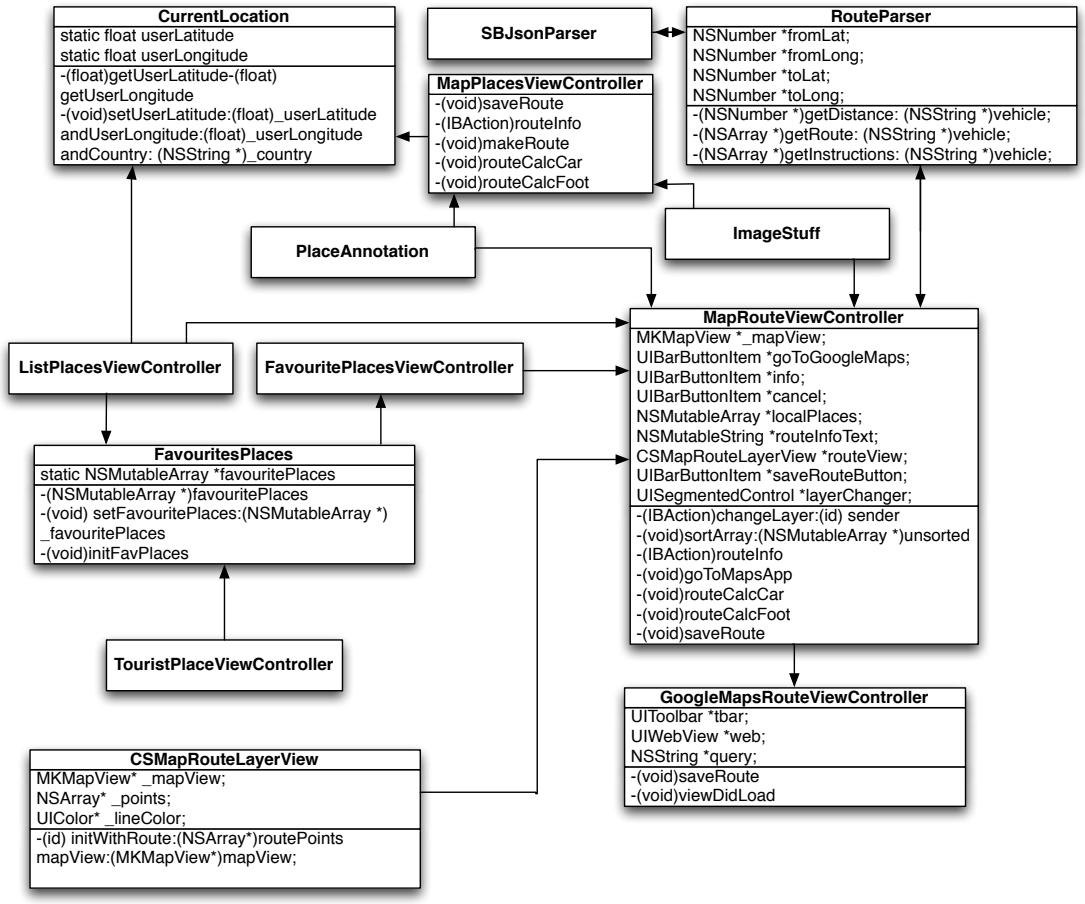

Figura 35: Diseño del controlador de rutas

La función principal de las distintas clases es:

- ***MapRouteViewController*:** Clase encargada de gestionar la ruta (óptima) correspondiente. Permite obtener las indicaciones paso a paso de la misma, guardarla como foto y cambiar la vista del mapa entre varios formatos (híbrido, satélite o mapa).

- ***GoogleMapsRouteViewController*:** Permite al usuario ver la ruta en la aplicación mapas del iPhone en el caso de que ésta contenga dos lugares. En el caso de que la ruta contenga más lugares, se cargará una vista Web (*UIWebView*) con la página de Google Maps y la ruta correspondiente.

- ***RouteParser***: Realiza la petición de ruta al servicio de CloudMade y pasar la información al parseador JSON correspondiente.

- ***SBJsonParser:*** Clase encargada de parsear el fichero JSON proveniente de la clase *RouteParser.*

- ***CSMapRouteLayerView:*** Contiene la vista con la ruta que se superpone sobre la vista del mapa de la clase *MapRouteViewController.*

- ***PlaceAnnotation***: Clase necesaria para encapsular los diferentes lugares como objetos para representar en el mapa (“chinchetas”).

- ***ImageStuff***: Gestiona varios aspectos de las imágenes como el tamaño y los bordes.

- ***FavouritePlaces*:** Guarda los lugares favoritos para que puedan ser accedidos desde cualquier parte del programa.

- ***CurrentLocation*:** Guarda la latitud y longitud del usuario para poder utilizarse en cualquier parte del programa.

Para las dos últimas clases, se decidió realizar un diseño basado en variables estáticos. Tanto la localización del usuario como los lugares favoritos necesitan ser consultados por diferentes clases dentro de la aplicación y la solución más cómoda es tener estos datos intactos a lo largo de la vida de la aplicación, haciendo uso por tanto de las características de los objetos estáticos.

Se puede comprobar como en este caso se han incluido los métodos relativos a la funcionalidad de rutas para la clase *MapPlacesViewController* en el diagrama de la Figura 35.

En la Tabla 15 se puede ver una explicación detallada de los atributos y métodos de la clase *MapRouteViewController.*

MapRouteViewController	
Atributos	
_mapView	Vista del mapa correspondiente.
goToGoogleMaps	Botón para cargar la ruta en la aplicación/web de Google Maps.
info	Botón para ver las indicaciones paso a paso.
localPlaces	Array con los lugares de la ruta.
routeView	Objeto de la clase *CSMapRouteLayerView* que actúa como capa con la ruta que se pinta sobre la vista *_mapView.*
saveRouteButton	Botón para guardar la ruta como imagen.
layerChanger	Cambia la vista del mapa (híbrido, satélite o mapa).
Métodos	
changeLayer	Método llamado al pulsar el botón *layerChanger.*
sortArray	Ordena los lugares por distancia para obtener la ruta óptima.
routeInfo	Método llamado al pulsar el botón *info.*
goToMapsApp	Método llamado al pulsar el botón *goToGoogleMaps.*
routeCalcCar	Calcula la ruta en el caso de realizarse en coche.
routeCalcFoot	Calcula la ruta en el caso de realizarse a pie.
saveRoute	Método llamado al pulsar el botón *saveRouteButton*.

Tabla 15: Clase MapRouteViewController

GoogleMapsRouteViewController	
Atributos	
tbar	Barra superior de la vista Web donde se encuentran, entre otros, los botones de para ir atrás y salvar la ruta.

GoogleMapsRouteViewController	
web	Objeto donde se encapsula la vista Web.
query	URL con la dirección de la ruta a mostrar.
Métodos	
saveRoute	Método encargado de guardar la ruta como imagen.
viewDidLoad	Método encargado de cargar la vista Web con la URL correspondiente.

Tabla 16: Clase GoogleMapsRouteViewController

3.1.4 Diseño del controlador del lugar turístico

Posiblemente la parte más importante de la aplicación, donde reside la mayoría de la funcionalidad. La obtención de lugares, fotos, opiniones y todas las acciones que se pueden realizar a partir de un lugar están contenidas en el diagrama de la Figura 36.

La función principal de las distintas clases es:

- ***TouristPlaceViewController*:** Una de las clases más importantes en la aplicación. Gestiona las acciones más importantes del lugar, esto es, ver las fotos, vídeos, opiniones y ver el lugar en el mapa. También permite llamar por teléfono al lugar, guardarlo en la agenda o compartir el lugar por mail o redes sociales.

- ***TouristicPlace*:** Clase que encapsula el objeto turístico. Destacar que aunque la clase contenga la palabra "*Touristic*", en realidad puede contener cualquier lugar genérico, turístico o no. Dicha clase es de suma importancia y contiene las características más importantes del lugar, como el nombre, dirección, teléfono... También contiene dos métodos para conseguir las opiniones, las imágenes y los vídeos como se detallará más adelante.

- ***GoogleSearchParser*:** Obtiene los diferentes lugares haciendo uso de la clase *SBJsonParser* para trabajar con el fichero JSON que se obtiene del API de Google Maps.

- ***GTMNSString+HTML*:** Encargada de eliminar los caracteres de escape típicos de HTML.

- ***PhotoTestController*:** Encargada de gestionar la visualización de las distintas fotos del lugar mediante el uso del API *Three20*.

- ***TouristPlaceVideosViewController*:** Mediante dicha clase, que hace uso del API *Three20,* el usuario puede ver los vídeos del lugar directamente desde la propia aplicación sin necesidad de apoyarse en la aplicación específica de YouTube del terminal.

- ***UserReviewsViewController*:** Gestiona la visualización de las opiniones por parte del usuario.

- ***TwitUpdateViewController*:** Se encarga de compartir el lugar seleccionado en la red social Twitter..

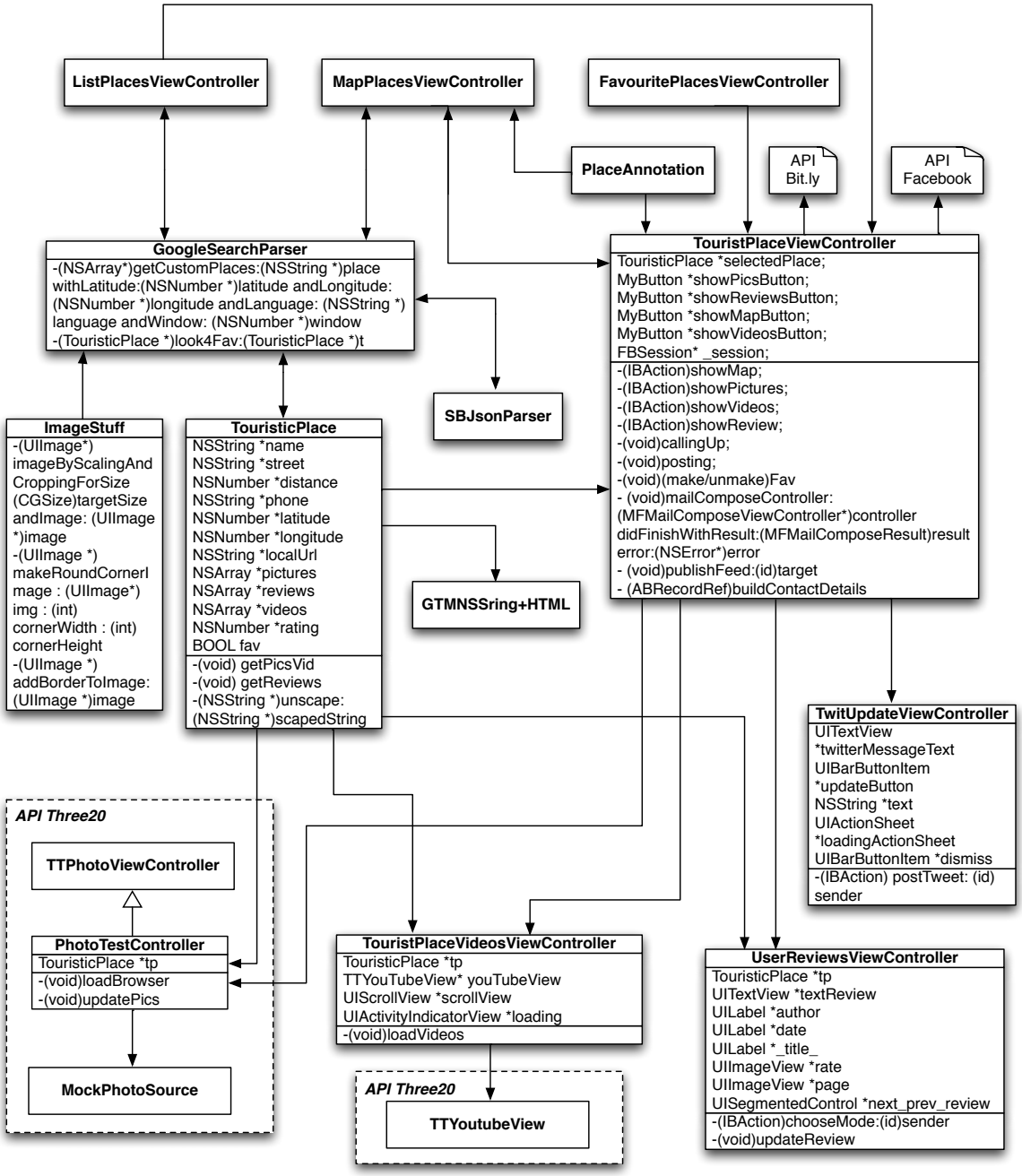

Figura 36: Diseño del controlador del lugar turístico

A continuación se detallarán algunos de los atributos y métodos más importantes de las clases del diagrama de la Figura 36.

GoogleSearchParser	
Métodos	
getCustomPlaces:withLatitud:andLongitude:andLanguage:andWindow	Método encargado de conseguir los diferentes lugares a través del API de Google Maps. Utiliza la clase *SBJsonParser* para trabajar con el fichero JSON obtenido.
look4Fav	Método encargado de comprobar que los lugares obtenidos por el método anterior no se encuentran entre los lugares favoritos guardados por el usuario. En caso contrario, activa el flag *fav* del lugar correspondiente.

Tabla 17: Clase GoogleSearchParser

TouristicPlace	
Atributos	
name	Nombre del lugar.
street	Dirección del lugar.
distance	Distancia desde la posición del usuario al lugar.
phone	Teléfono del lugar.
latitude	Latitud de la posición del lugar.
longitude	Longitud de la posición del lugar.
localUrl	Web del lugar en Google Maps.
pictures	Array con las URL de las fotos del lugar.
reviews	Array con las opiniones del lugar.
videos	Array con las URL de los vídeos.
rating	Puntuación del lugar.
fav	Valor booleano que indica si el lugar se encuentra entre los favoritos del usuario.

TouristicPlace	
Métodos	
getPicsVid	Método que se ejecuta en un hilo para conseguir las imágenes y vídeos del lugar.
getReviews	Método que se ejecuta en un hilo para conseguir las opiniones del lugar.
unescape	Método utilizado por *getPicsVid* y *getReviews* para eliminar los caracteres de escape de HTML.

Tabla 18: Clase TouristicPlace

TouristPlaceViewController	
Atributos	
selectedPlace	Lugar seleccionado por el usuario.
showPicsButton	Botón encargado de mostrar las imágenes del lugar.
showReviewsButton	Botón encargado de mostrar las opiniones del lugar.
showMapButton	Botón encargado de mostrar el lugar en el mapa.
showVideosButton	Botón encargado de mostrar los vídeos del lugar.
Métodos	
showMap	Método llamado al pulsar sobre el botón *showMapButton.*
showPictures	Método llamado al pulsar sobre el botón *showPicsButton.*
showVideos	Método llamado al pulsar sobre el botón *showVideosButton.*
showReview	Método llamado al pulsar sobre el botón *showReviewsButton.*
callingUp	Método encargado de llamar por teléfono al lugar.
posting	Método encargado de gestionar la publicación del lugar en las redes sociales o de enviarlo por email.

TouristPlaceViewController	
(make/unmake)Fav	Guarda/elimina el lugar como favorito.
mailComposeController:didFinishWithResult:error	Método encargado de realizar el envío final del email con el lugar seleccionado.
publishFeed	Método que realiza la publicación final del lugar seleccionado en Facebook.
buildContactDetails	Guarda el lugar seleccionado en la agenda.

Tabla 19: Clase TouristPlaceViewController

TouristPlaceVideosViewController	
Atributos	
tp	Lugar sobre el que se muestran los vídeos
youTubeView	Objeto perteneciente el API *Three20* a partir del cual se consigue la visualización del vídeo dentro de la propia aplicación.
Métodos	
loadVideos	Método encargado de cargar los vídeos en la vista actual (hasta un máximo de seis).

Tabla 20: Clase TouristPlaceVideosViewController

PhotoTestController	
Atributos	
tp	Lugar sobre el que se muestran los imágenes
loadBrowser	Método encargado de cargar el visualizador de imágenes, perteneciente al API *Three20,* utilizado en la aplicación.
Métodos	
updatePics: método	Método encargado de realizar diversas acciones sobre las imágenes para después lanzar un hilo con el método *loadBrowser*.

Tabla 21: Clase PhotoTestController

UserReviewsViewController	
Atributos	
tp	Lugar sobre el que se muestran las opiniones.
textReview	Texto de la opinión del lugar.
author	Autor de la opinión del lugar.
date	Fecha de la opinión del lugar.
title	Título de la opinión del lugar.
rate	Imagen con la puntuación de la opinión del lugar.
page	Imagen que identifica la página Web de la que proviene la opinión (se han elegido las nueve páginas más comunes).
next_prev_review	Botón para avanzar o retroceder de opinión.
Métodos	
chooseMode	Método llamado al pulsar sobre el botón *next_prev_review.*
updateReview	Método llamado por *chooseMode* encargado de actualizar la opinión

Tabla 22: Clase UserReviewsViewController

Un aspecto importante a destacar en el diseño de dicha parte es la obtención de las imágenes, fotos y opiniones. El objeto *TouristicPlace* es creado a partir de la clase *GoogleSearchParser* pero en este momento no posee ni fotos, ni vídeos ni opiniones, simplemente las características más básicas del mismo (nombre, dirección...). Esto se ha realizado de esta manera para que el usuario obtenga la lista de lugares lo más rápido posible. En muchos casos el usuario puede simplemente necesitar ubicar el sitio en el mapa o ver la dirección, por lo que hacer esperar al usuario por la obtención de las características multimedia que no va a utilizar es un precio bastante caro a pagar. La solución por la que se ha optado es por el uso de hilos. En el momento en el que el usuario pulsa sobre el lugar para ir a la vista principal del lugar, se lanzan los hilos encargados de conseguir la información que falta. Esto se puede ver gráficamente en la Figura 37.

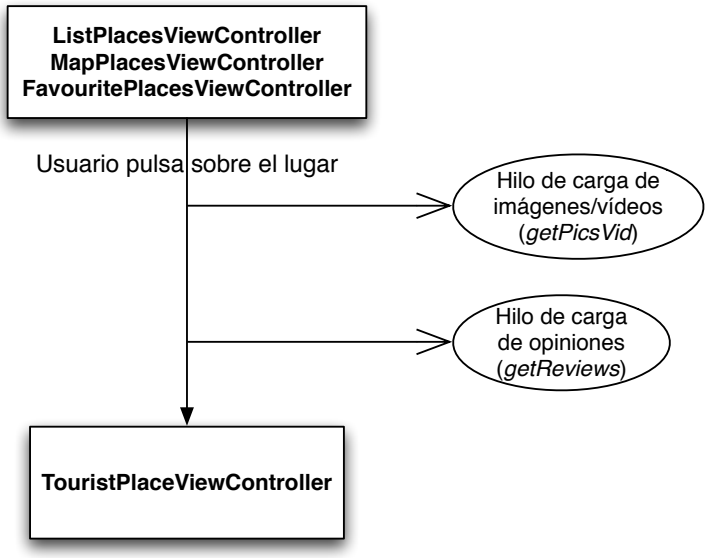

Figura 37: Hilos de carga de información multimedia

Los métodos *getPicsVid* y *getReviews* pertenecen a la clase *TouristicPlace,* por lo que de alguna manera, es el propio objeto el que se autocompleta con los datos que faltan. En una situación normal de uso por parte del usuario (donde no pulsa inmediatamente en el botón de imágenes u opiniones), se consigue que la espera sea prácticamente nula y sea todo transparente para el usuario, obteniendo la información que necesita de forma instantánea y sin esperas.

Además de las clases mencionadas anteriormente, el diagrama de la Figura 36 muestra dos API utilizadas:

- **API de Facebook**: Permite compartir el lugar en la red social más grande del mundo. Usa OAuth para garantizar la seguridad del usuario. La publicación incluye un mensaje personal, así como la foto del lugar y la URL a la web de Google Maps correspondiente.

- **API de Bit.ly**: La característica innata de Twitter, servicio de microblogging en el que las publicaciones no pueden contener más de 160 caracteres, promovió el uso de dicho servicio de acortamiento de URL. Su funcionamiento es idéntico al de otras API usadas en el proyecto. Tras un previo registro para obtener la clave necesaria, una petición HTTP incluyendo la misma da como resultado un fichero JSON del cual se obtiene la URL acortada.

3.2 Diagramas de secuencia

3.2.1 Inicio de la aplicación

En la Figura 38 se puede ver el diagrama de secuencia del inicio de la aplicación, es decir, la búsqueda de lugares turísticos cercanos al usuario. Notar que se ha omitido todo lo relacionado con la elección de idioma (que sólo se da la primera vez que se abre la aplicación) y las clases internas del SDK participantes en el inicio de cualquier aplicación iPhone.

Mencionar que el objeto *quartz_core_loading_view* del diagrama es simplemente una vista semi-transparente, utilizando el API de *QuartzCore*[18] del SDK para alertar al usuario de que el programa está obteniendo datos.

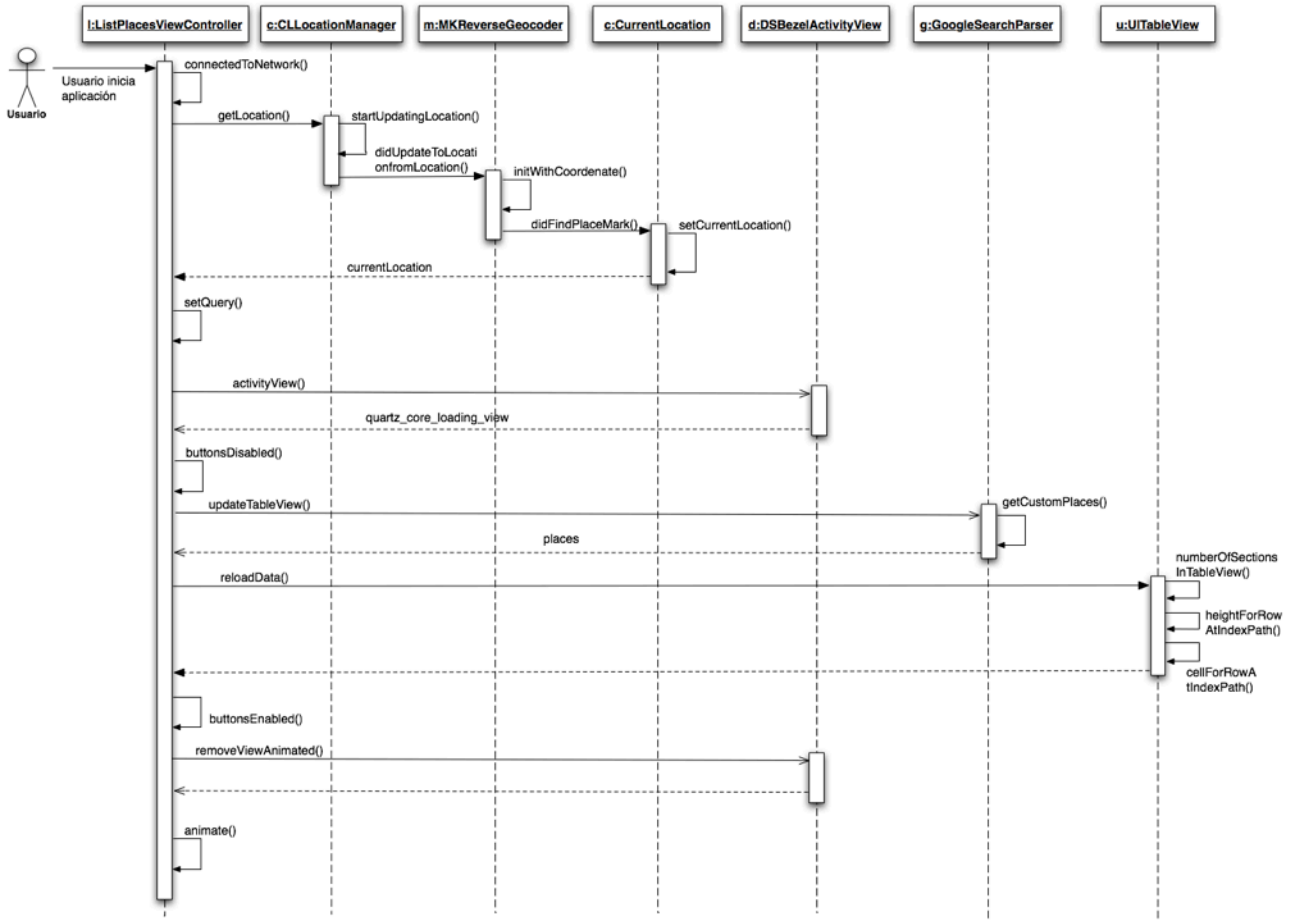

Figura 38: Diagrama de secuencia - inicio de la aplicación

[18] Si se desea más información, véase [30].

3.2.2 Búsqueda de lugares cualesquiera cercanos

El siguiente diagrama de secuencia muestra el flujo de información existente cuando el usuario realiza una búsqueda de un lugar genérico cercano a su posición. Como se puede ver, en este caso el usuario realiza dos acciones:

1. Pulsar en el botón para realizar la búsqueda.

2. Aceptar la misma tras introducir la información necesaria.

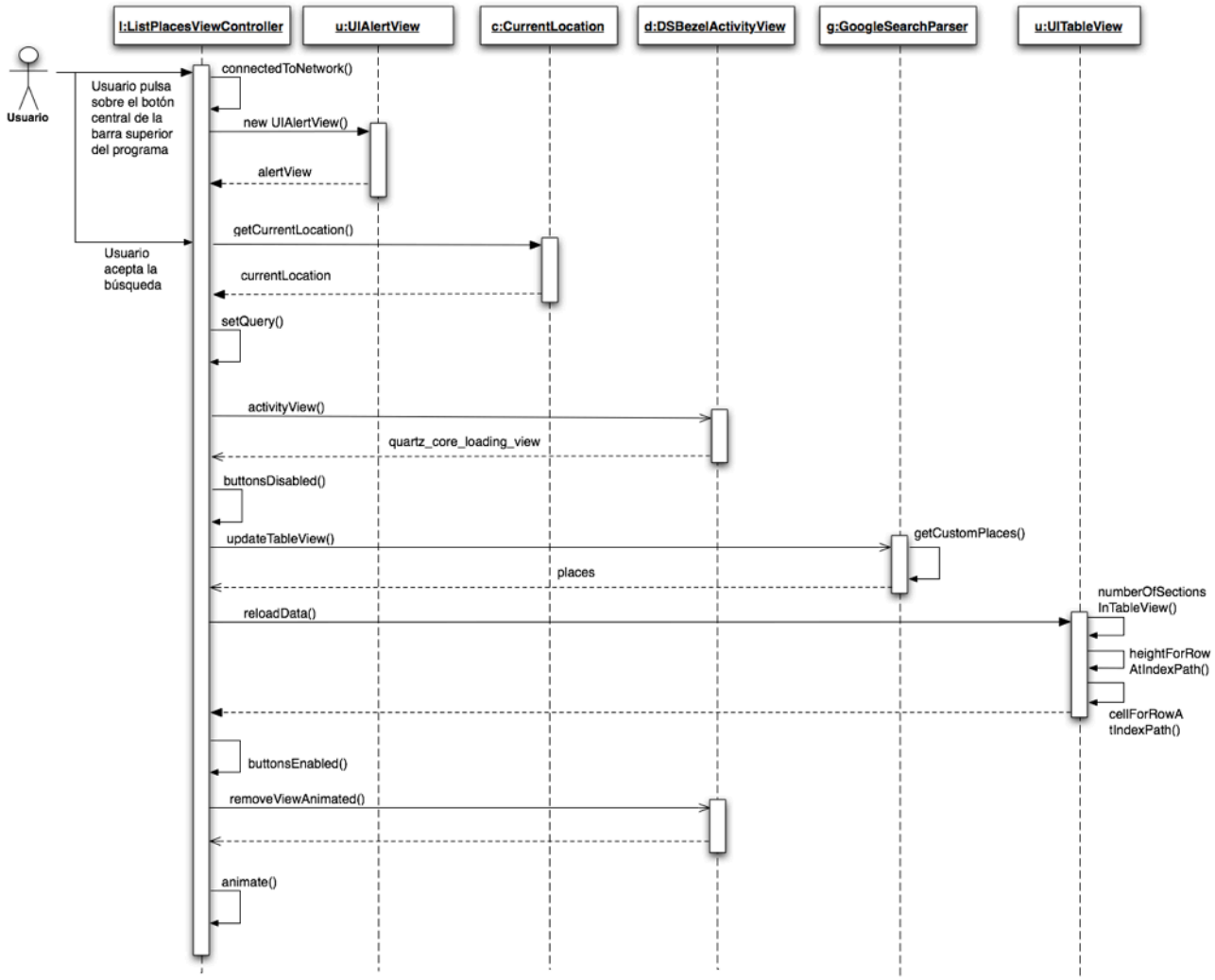

Figura 39: Diagrama de secuencia - búsqueda de lugares cercanos

3.2.3 Ver un lugar y búsqueda de fotos y opiniones

El siguiente diagrama de secuencia muestra los mensajes que se intercambian los distintos objetos de la aplicación cuando el usuario pulsa sobre un lugar cualquiera para ver más información sobre el mismo.

Como se explicó previamente, y como se puede ver en la Figura 40, en el momento en el que se carga la nueva vista del lugar, automáticamente se ejecutan dos hilos encargados de obtener la información relativa a fotos, vídeos y opiniones.

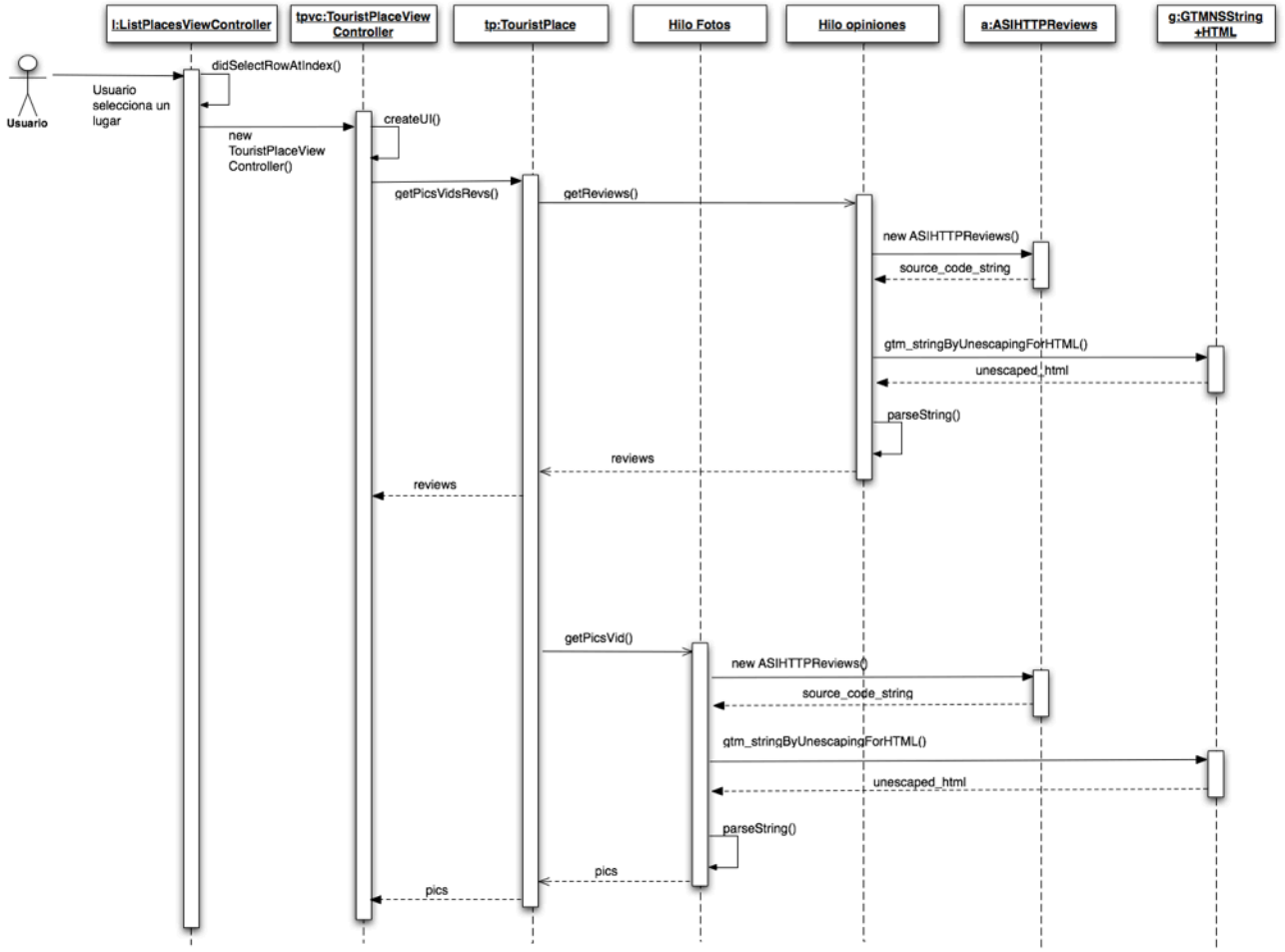

Figura 40: Diagrama de secuencia - ver un lugar

3.2.4 Visualización de fotos

En la Figura 41 se puede ver la secuencia de acciones que ocurren en el momento en el que el usuario pulsa el botón encargado de ejecutar el visualizador de imágenes. Notar como el objeto de la clase *TouristicPlace,* en el caso más general, tiene ya almacenadas las fotos obtenidas en el diagrama de secuencia anterior, por lo que la carga del visualizador es prácticamente instantánea.

Destacar que las clases *MockPhotoSource* y *TTPhotoSource* pertenecen al API *Three20* que se comentará más en detalle en el capítulo 4.

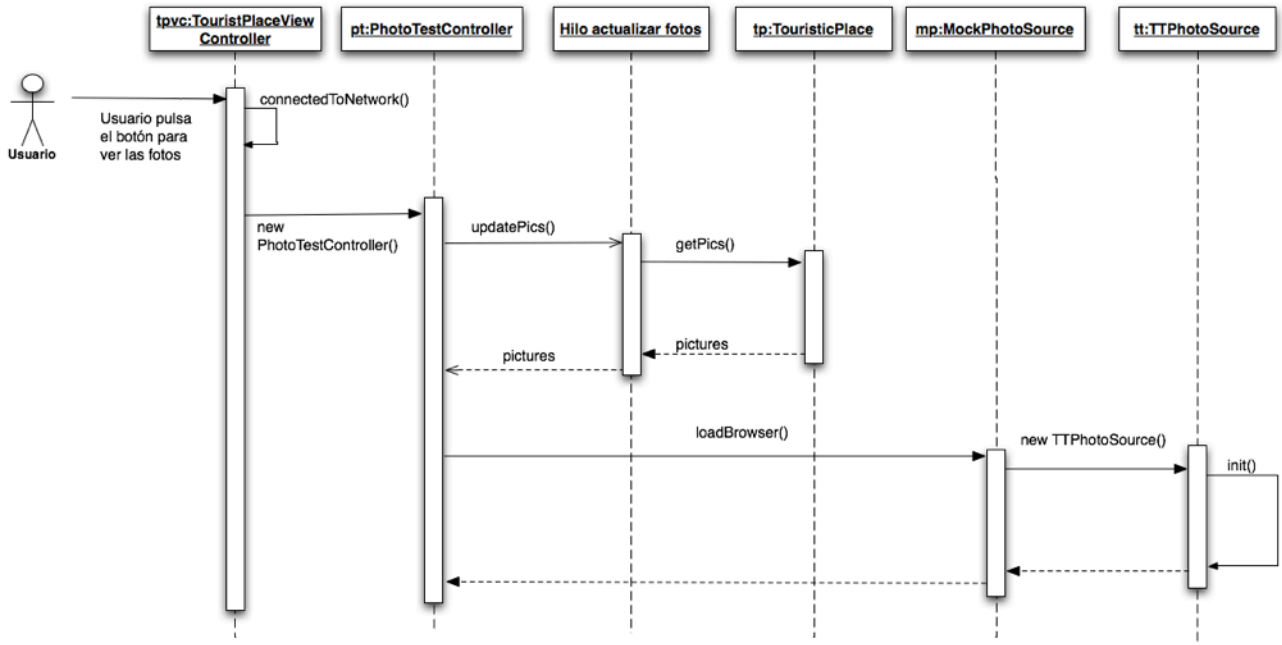

Figura 41: Diagrama de secuencia - visualización de fotos

3.2.5 Visualización de opiniones

Al igual que en el caso anterior, en el diagrama de la Figura 42 se puede ver la interacción de los objetos cuando el usuario pulsa el botón para ver información sobre las opiniones del lugar.

Como se comentó previamente, el objeto *TouristicPlace* tiene ya almacenadas las opiniones, por lo que el tiempo de carga es ínfimo.

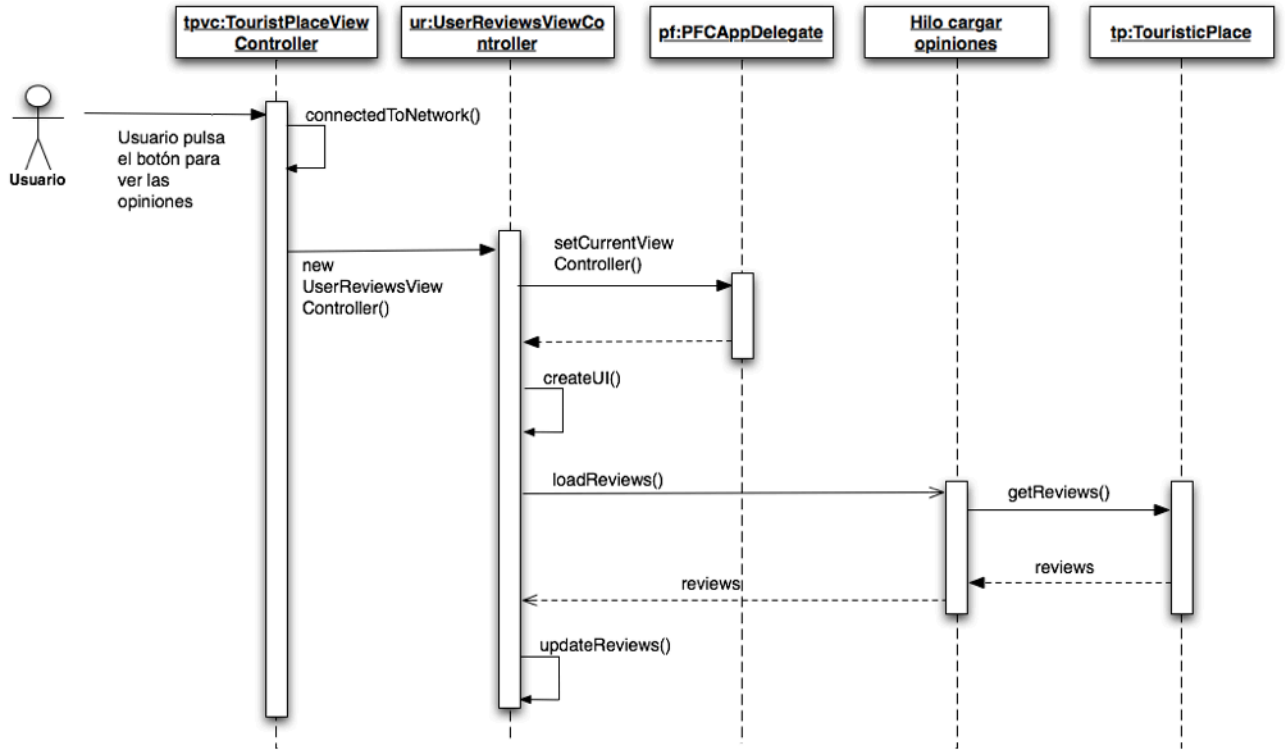

Figura 42: Diagrama de secuencia - visualización de opiniones

3.2.6 Cálculo de rutas

En la Figura 43 se puede ver uno de los aspectos claves de la aplicación, esto es, el cálculo de rutas entre un lugar arbitrario de lugares.

Se puede ver como la interacción del usuario en este caso es doble:

1. Por un lado debe seleccionar el botón encargado de iniciar la gestión de rutas (disponible en la vista correspondiente de los controladores *ListPlacesViewController* y *FavouritePlacesViewController*).

2. A continuación debe seleccionar los lugares que se desean incluir en la ruta para después aceptar la misma (recibiendo dicha selección de lugares a través del método *indexPathForSelectedRows*).

Se puede también la utilización de la clase SBJsonParser, encargada de obtener la información del fichero Json proveniente del servicio CloudMade. Destacar por último la utilización de la clase *CSMapRouteLayerView* que es la ruta que se superpone sobre el mapa de la aplicación (capa sobre el objeto *MKMapView*).

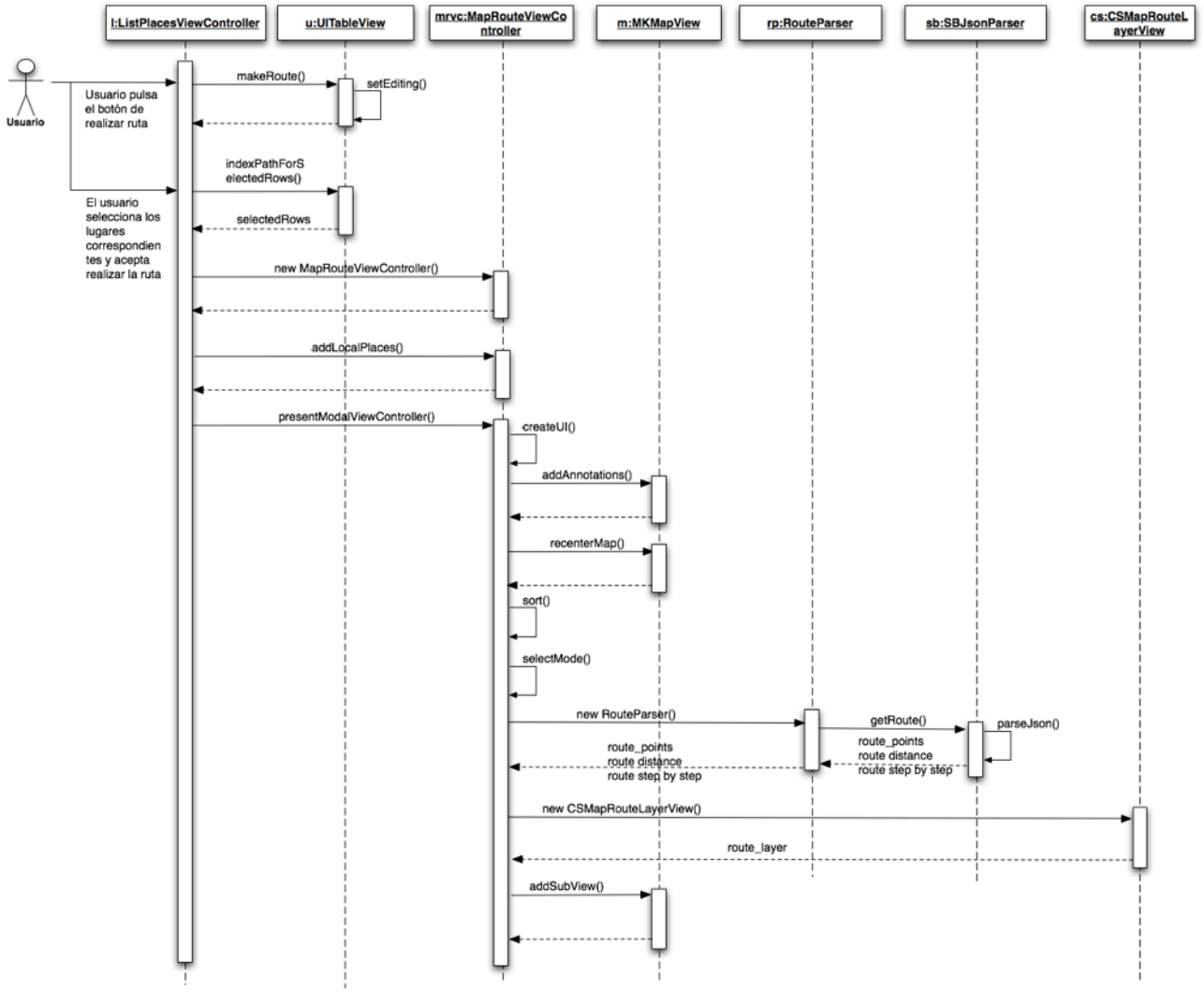

Figura 43: Diagrama de secuencia - cálculo de rutas

3.2.7 Compartir en Facebook

En el siguiente diagrama se puede ver que, de nuevo, la interacción del usuario para compartir un lugar en la red social Facebook requiere de dos pasos:

1. Seleccionar el botón existente en la vista encargado de compartir el lugar (por correo o en las redes sociales Facebook y Twitter).

2. Seleccionar la opción deseada (Facebook en este caso)

En este caso se ha utilizado el API proporcionada por esta red social (FBConnect), a la que pertenecen las clases *FBSession, FBRequest y FBLoginDialog* que se pueden ver en el diagrama.

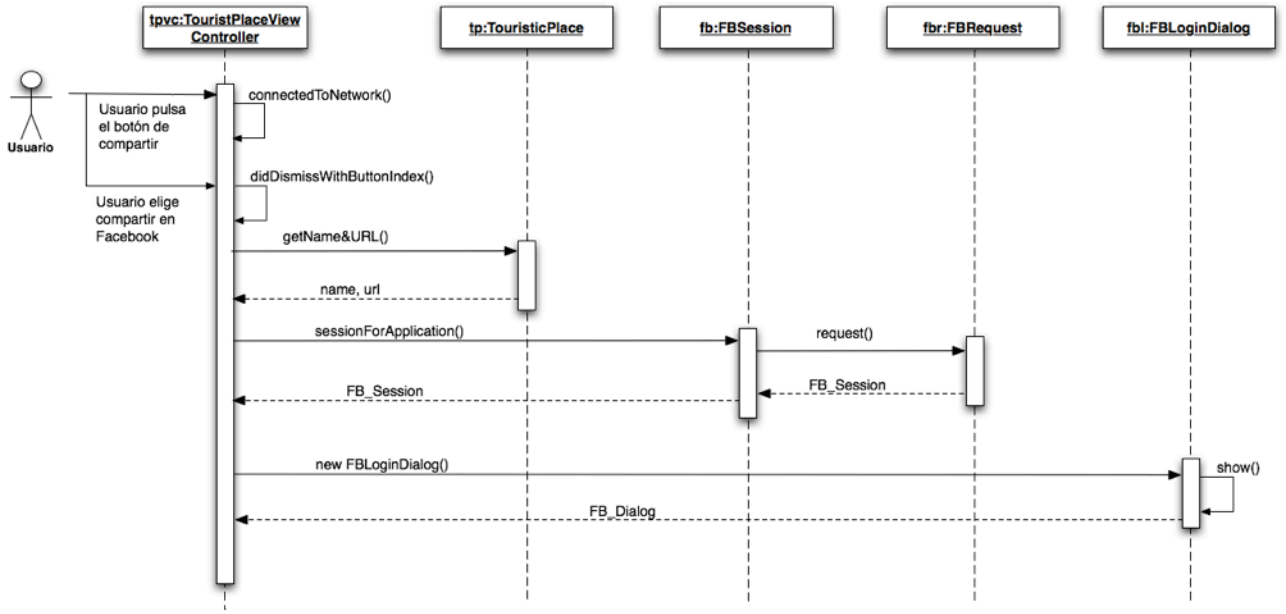

Figura 44: Diagrama de secuencia - compartir en Facebook

3.2.8 Guardar favorito

Por último, la gestión de favoritos usa distintos aspectos importantes dentro de cualquier aplicación iPhone.

Por un lado se puede ver como se trabaja con dos clases pertenecientes al SDK del sistema como son *UINavigationBar* y *UITabBarController*. La primera ha sido utilizada para cambiar el icono de favorito para representar que dicho lugar ha sido elegido como tal. La segunda es utilizada para conseguir cambiar el *badge* (pequeño icono rojo con el número de favoritos guardados) de dicha pestaña.

Otro aspecto importante es la utilización de la clase *NSUserDefaults* con la que se consigue guardar el lugar favorito de forma permanente para que esté disponible a lo largo de las distintas sesiones del programa.

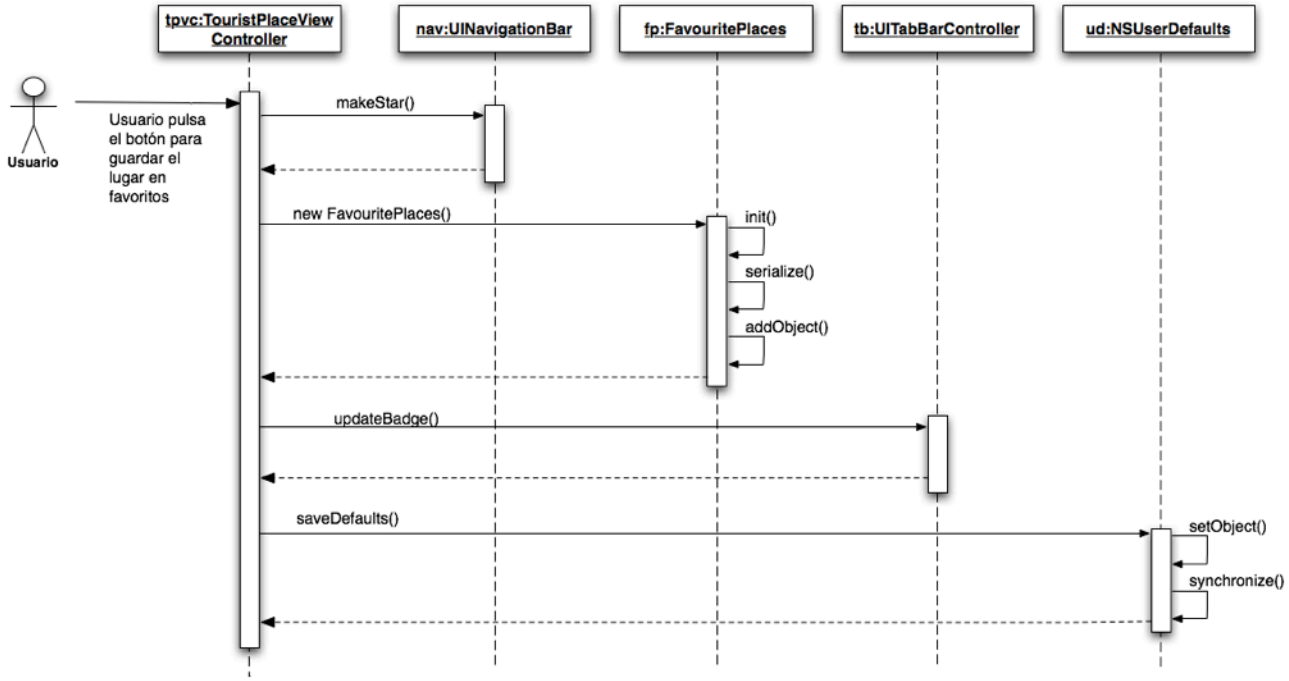

Figura 45: Diagrama de secuencia - guardar favoritos

Capítulo 4: Implementación

En este quinto capítulo del proyecto, que versa sobre la implementación del software desarrollado, se tratarán dos partes muy diferenciadas. Por un lado se detallarán los aspectos más importantes y relevantes de las distintas partes del código desarrollado así como se proporcionarán los resultados satisfactorios del plan de pruebas efectuado en el capítulo 2.

4.1 Aspectos de la implementación

En esta sección se adentrará en la implementación de algunas partes importantes de la aplicación, explicándose parte del código que resulta de especial interés.

4.1.1 Localización del usuario

A continuación se analizarán las partes más importantes del Framework *CoreLocation,* el cual gestiona todo lo relacionado con el cálculo del posicionamiento del usuario.

4.1.1.1 Modo de operación

Su funcionamiento se basa en dos clases y un protocolo.

- **Clases**: *CLLocationManager, CLLocation*
- **Protocolo**: *CLLocationManagerDelegate*

El funcionamiento a grandes rasgos consiste en:

1. Crear una instancia de la clase *CLLocationManager*
2. Asignar a la anterior clase el protocolo *CLLocationManagerDelegate*

Los dos métodos que gestiona el protocolo para la gestión de eventos son:

```
-(void)locationManager:(CLLocationManager*)manager
didUpdateToLocation:(CLLocation*)newLocation
fromLocation:(CLLocation*)oldLocation;
```

```
- (void)locationManager:(CLLocationManager*)manager
didFailWithError:(NSError*)error;
```

El primero de ellos es llamado siempre que se detecte algún cambio de posición, guardándose en la variable "*newLocation*" la posición del terminal en ese justo momento.

Concretando más, la forma de acceder a la latitud y longitud exactas sería:

```
double lat = newLocation.coordinate.latitude;
double lon = newLocation.coordinate.longitude;
```

El segundo método, como se puede deducir de su nombre, gestiona los errores ocurridos a la hora del cálculo de la posición. Lo normal es que dicho método gestione algún tipo de alerta para avisar al usuario de que algo ha ocurrido mal.

A continuación se comentarán otros detalles del framework de localización.

4.1.1.2 Precisión

CoreLocation permite al desarrollador establecer el nivel de precisión que se desea tener en la aplicación. Según esto, el terminal elegirá uno de los tres métodos de localización mencionados en 2.2.5 (GPS, WPS y *Cell ID*). La forma de establecer la precisión sería la siguiente:

```
CLLocationManager* locManager = [[CLLocationManager alloc] init];
locManager.desiredAccuracy = kCLLocationAccuracyBest;
```

Exiten muchas variables para determinar la precisión que se busca tener en la aplicación, entre ellas *kCLLocationAccuracyNearestTenMeters*, *kCLLocationAccuracyKilometer,* aunque normalmente se suele usar *kCLLocationAccuracyBest* y dejar que sea el dispositivo el que elija la mejor precisión según las circunstancias en las que se encuentre el usuario (ausencia de señal GPS, ausencia de puntos de acceso wifi cercanos...).

Hay que tener en cuenta que cuanta mayor precisión se requiera, más batería requerirá el proceso. Esto es algo muy importante a la hora de diseñar software para dispositivos móviles con una vida útil muy limitada en algunos casos.

4.1.1.3 Resolución

El siguiente parámetro importante a analizar es la resolución que tiene el sistema (también conocido como filtro de distancia). Se trata simplemente de indicar al sistema la distancia mínima (en metros) que se debe mover el usuario para que se produzca el evento de actualización de la posición. Éste parámetro se indica mediante la variable *distanceFilter.*

```
CLLocationManager* locManager = [[CLLocationManager alloc] init];
locManager.distanceFilter = 3000;
```

4.1.1.4 Puesta en marcha y apagado de la localización

Para encender y apagar la capacidad de localización del terminal, simplemente es necesario crear una instancia del protocolo *CLLocationManager* para llamar respectivamente a los métodos *startUpdatingLocation* y *stopUpdatingLocation.*

```
CLLocationManager* locManager = [[CLLocationManager alloc] init];
locManager.delegate = self;
[locManager startUpdatingLocation];
```

```
CLLocationManager* locManager = [[CLLocationManager alloc] init];
locManager.delegate = self;
[locManager stopUpdatingLocation];
```

4.1.1.5 Geocodificación inversa

Destacar que, tras numerosas pruebas con la aplicación, se decidió implementar geocodificación inversa [29]. Ésta consiste en, a través de la latitud y la longitud del usuario, encontrar información del área en la que se encuentra, como el país, la ciudad o la calle en última instancia. Esta elección está basada en el hecho de que google proporciona resultados más precisos cuando la búsqueda se encuentra en el idioma nativo del lugar del cuál se quieren obtener los lugares.

Supóngase que se realiza un viaje desde Madrid a Nueva York y desea obtener los lugares turísticos cercanos. La aplicación por defecto tendrá el idioma español y por tanto, sin geocodificación inversa, se realizaría la búsqueda según el idioma nativo del usuario:

```
http://ajax.googleapis.com/ajax/services/search/local?v=1.0&q=lugares+de
+interés+turístico&near=40.714353,-74.005973&rsz=large&start=0&hl=es
```

En cambio, con geocodificación inversa, lo primero que se realizar es el cálculo del país en el que se encuentra el usuario según la latitud y longitud calculadas. A partir de este dato, la petición se realiza según el idioma del país donde se encuentre el terminal. En la aplicación se han tenido en cuenta cinco (EEUU, Alemania, Francia, Italia, ESpaña) . La petición por tanto sería la siguiente.

```
http://ajax.googleapis.com/ajax/services/search/local?v=1.0&q=tourist
attractions&near=40.714353,-74.005973&rsz=large&start=0&hl=es
```

Notar como el parámetro hl sigue estando en el idioma del usuario y en el caso de existir opiniones en el idioma nativo del mismo, éstas serán las primeras que se mostrarán.

A continuación se puede ver parte del método implementado que se ejecuta una vez el *Framework* de localización ha efectuado correctamente la geocodificación inversa. Notar como en primer lugar se calcula el país en el que se encuentra el usuario para posteriormente construir la petición en base a él.

```
- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder didFindPlacemark:(MKPlacemark *)
placemark{

        country=placemark.countryCode;
        ......

        if ([country isEqualToString:@"US"]) {
                query=[[NSString alloc] initWithFormat:@"category: tourist attraction near %f,%f",
userLatitude, userLongitude];

        }
        else if([country isEqualToString:@"ES"]){

                query=[[NSString alloc] initWithFormat:@"categoria: lugares de interes turístico
near %f,%f", userLatitude, userLongitude];
        }
....

}
```

4.1.2 Autenticación básica en Twitter

Aunque Twitter ha cambiado recientemente sus sistema de autenticación a *OAuth*, es interesante analizar el código utilizado para conseguir una autenticación básica[19] que funcionó perfectamente durante muchos meses de la vida del proyecto y permitió compartir lugares en dicha red social.

Sólo se mencionarán los métodos importantes y las partes relevantes dentro de los mismos.

El código que se ejecuta para compartir algo en Twitter es el siguiente:

[19] Se entiende por autenticación básica aquella basada en usuario y contraseña.

```
TwitterRequest *twitReq=[TwitterRequest new];

[twitReq statuses_update:twitterMessageText.text delegate:self requestSelector:@selector
(status_updateCallback:)];
```

La variable *twitterMessageText.text* contiene el texto que se desea compartir. Para ello se incluye como parámetro en el método *status_update* que se ve a continuación.

```
-(void)statuses_update:(NSString *)status delegate:(id)requestDelegate requestSelector:(SEL)
requestSelector; {

        NSURL *url = [NSURL URLWithString:@"http://twitter.com/statuses/update.xml"];
        requestBody = [NSString stringWithFormat:@"status=%@",status];
        [self request:url];
}
```

En este método se realizan dos acciones importantes:

1. Indicar a Twitter que se quiere realizar una actualización de estado mediante la llamada al método *request* con la url http://twitter.com/statuses/update.xml.

2. Construir la variable *requestBody* que contiene tal actualización de estado.

A continuación se pasará a analizar el método *request* que contiene partes destacadas de la implementación.

```
-(void)request:(NSURL *) url {
        ...
        theRequest   = [[NSMutableURLRequest alloc] initWithURL:url];
        [theRequest setHTTPMethod:@"POST"];
        ....
        receivedData=[[NSMutableData data] retain];
}
```

El método recibe como parámetro una URL (en concreto, *http://twitter.com/statuses/update.xml*). Las acciones más importantes que se realizan son:

1. Se crea el objeto *NSMutableURLRequest* que consiste en una clase de Objective-C mediante la cual se encapsulan URLs para hacer peticiones Web.

2. Se indica que tal petición va a ser del tipo “POST” y se introduce la actualización de estado (*requestBody*) dentro de la misma.

3. Se crea la conexión URL con la petición anterior y se crea la variable *receivedData* donde se guardará la información recibida de los servidores de Twitter en respuesta a la petición.

El siguiente método se llama cada vez que el servicio Web con el que se intenta comunicar requiere autenticación basada en usuario y contraseña.

```
- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:
(NSURLAuthenticationChallenge *)challenge {

        NSURLCredential * newCredential=[NSURLCredential credentialWithUser:[self
        username] password:[self password] persistence:NSURLCredentialPersistenceNone];

        [[challenge sender] useCredential:newCredential forAuthenticationChallenge:challenge];
        }
}
```

Las dos acciones importantes efectuadas son:

1. Se crea mediante la clase *NSURLCredential* las credenciales necesarias para establecer la correcta comunicación con Twitter. Para ello se crea un objeto de la clase anterior con el nombre de usuario y contraseña para la autenticación.

2. Se envía dicho credencial de nuevo al servidor mediante la variable *challenge* (reto) que se recibe como argumento del método.

El último método creado se llama *connectionDidFinishLoading* y, como su propio nombre indica, se llama cuando se han terminado de recibir datos del servidor.

En dicho método, simplemente se llama a una rutina llamada *callback*, pasándole como objeto los datos recibidos desde Twitter para realizar las acciones oportunas. También se realizan acciones de liberación de memoria.

4.1.3 Gestión de las imágenes

Uno de los aspectos importantes de la implementación fue la gestión de las imágenes obtenidas. Debido a que se quiso dar una apariencia visual mejorada a las mismas, se implementó una clase llamada *ImageStuff* con tres métodos en ella. Se realizan algunos tratamientos básicos sobre las imágenes a través de distintas funciones proporcionadas por el *Framework QuartzCore* [30] disponible en el SDK del iPhone.

En cada uno de los métodos se proporcionará una pequeña parte del código para hacer una idea al lector del uso del *Framework* utilizado, para posteriormente explicar mediante palabra todas las acciones realizadas en el mismo.

El primero de ellos se llama *imageByScalingAndCroppingForSize* y redimensiona la imagen al tamaño indicado. Se puede ver a continuación:

```
- (UIImage*)imageByScalingAndCroppingForSize:(CGSize)targetSize andImage: (UIImage *)image;
{
        ......
        scaledWidth  = width * scaleFactor;
        scaledHeight = height * scaleFactor;
        thumbnailPoint.y = (targetHeight - scaledHeight) * 0.5;
        thumbnailPoint.x = (targetWidth - scaledWidth) * 0.5;
        .....
        thumbnailRect.origin = thumbnailPoint;
        thumbnailRect.size.width  = scaledWidth;
        thumbnailRect.size.height = scaledHeight;
        [sourceImage drawInRect:thumbnailRect];
        newImage = UIGraphicsGetImageFromCurrentImageContext();

        return newImage;
}
```

Las acciones que se realizan en el método se pueden dividir en tres bloques:

1. Primero se obtienen distintos datos de la imagen que recibe como parámetro y el parámetro *targetSize* que supone el tamaño al que se quiere transformar.

2. Se calculan los distintos factores de escalado (*scaledWidth, scaledHeight*) y el nuevo recuadro para la imagen reescalada (*thumbnailRect.origin, thumbnailRect.size.width*, *thumbnailRect.size.height*).

3. Se dibuja la nueva imagen en el cuadro anterior, se guarda el contexto y se devuelve la misma.

El siguiente método se llama *makeRoundCornerImage* y, como indica su nombre, suaviza los bordes de la imagen redondeándolos.

```
-(UIImage *)makeRoundCornerImage : (UIImage*) img : (int) cornerWidth : (int) cornerHeight
{
        .....
        CGContextMoveToPoint(context, fw, fh/2);
        CGContextAddArcToPoint(context, fw, fh, fw/2, fh, 1);
        CGContextAddArcToPoint(context, 0, fh, 0, fh/2, 1);
        .......
        CGContextDrawImage(context, CGRectMake(0, 0, w, h), img.CGImage);
        CGImageRef imageMasked = CGBitmapContextCreateImage(context);
        newImage = [[UIImage imageWithCGImage:imageMasked] retain];

        return [newImage autorelease];

}
```

El funcionamiento del método es bastante simple. En primer lugar se guarda contexto[20] de la imagen para posteriormente modificar éste mediante las variables *cornerWidth* y *cornerHeight* a través del método *CGContextAddArcToPoint.* Una vez se ha realizado este proceso, se crea la nueva imagen basada en el contexto modificado de la imagen original.

El último método es *addBorderToImage* y ha sido utilizado para dotar a la imagen de un borde del color y grosor que se desee.

```
-(UIImage *)addBorderToImage:(UIImage *)image {
        .......
        CGContextDrawImage(ctx, CGRectMake(0, 0, (CGFloat)width, (CGFloat)height), bgimage);

        CGContextSetStrokeColorWithColor(ctx, [UIColor darkGrayColor].CGColor);
        CGFloat borderWidth = (float)width*0.04;
        CGContextSetLineWidth(ctx, borderWidth);
        .......
        CGImageRef cgimage = CGBitmapContextCreateImage(ctx);
        UIImage *newImage = [UIImage imageWithCGImage:cgimage];

        return newImage;
}
```

Como se puede ver, el grosor del borde es controlado mediante la variable *borderWidth* y se ha elegido que éste sea un 4% de la anchura de la imagen. El color se elige mediante el método *CGContextSetStrokeColorWithColor* donde se ha elegido el color gris oscuro (*darkGrayColor*).

[20] Un contexto de una imagen en *QuartzCore* se compone de todas las características de la imagen en un instante dado. Si se desea más información, véase [30].

4.1.4 Framework *Three20*

Three20 [31] es un *Framework* de código libre independiente del SDK del iPhone. Éste ha sido desarrollado por el creador de la aplicación de Facebook para iPhone, el conocido desarrollador Joe Hewitt, y proporciona muchas mejoras sobre el SDK original (nuevas listas, etiquetas, mecanismos asíncronos de comunicación web, visor de imágenes...).

En el ámbito del presente proyecto, se ha utilizado simplemente para implementar el visor de imágenes donde se obtienen las miniaturas de las mismas con unos efectos y transiciones muy visuales.

A continuación se dará una breve noción de los dos métodos utilizados para generar el visor con el Framework *Three20*. El primer método, llamado *updatePics,* es el encargado de almacenar las fotos en el visor.

```
-(void)updatePics{

        .....

        if([thumb rangeOfString:@"panoramio"].length!=0){

                normal=[thumb stringByReplacingOccurrencesOfString:@"static"
withString:@"www"];
                normal=[normal stringByReplacingOccurrencesOfString:@"iw-thumbnail"
withString:@"medium"];
        }
        else {
                normal=thumb;
        }

                        [pics addObject:[[[MockPhoto alloc] initWithURL:normal smallURL:thumb
size:CGSizeZero] autorelease]];

        .......
}
```

Destacar varios aspectos:

1. La variable *normal* almacena la foto en su tamaño estándar, mientras que la variable *thumb* almacena la miniatura. El método encargado de parsear las fotos de la página web proporciona las miniaturas. Por tanto en dicho método se trasforma la URL (de Panoramio normalmente) para conseguir la misma en una resolución superior.

2. Una vez se tienen las URL tanto de la miniatura como de la foto en resolución normal, se crean objetos de la clase *MockPhoto* (perteneciente a Three20) y se introducen en el array *pics.*

```
[pics addObject:[[[MockPhoto alloc]
initWithURL:normal smallURL:thumb size:CGSizeZero]
autorelease]];
```

El método *loadBrowser* es el encargado de lanzar el visor con las imágenes cargadas en el método anterior. Las dos acciones principales son:

1. Alertar al usuario en el caso de que no exista ninguna foto para el lugar.

2. Cargar el visor mediante la creación de un objeto de la clase *MockPhotoSource* que se almacena en la variable *photoSource*.

```
-(void)loadBrowser{

        if([[tp pictures] count]==0){
                        [self performSelector:@selector(showAlert) withObject:nil afterDelay:0];
        }
        else {
                .....

                self.photoSource=[[[MockPhotoSource alloc] initWithType:MockPhotoSourceNormal
title:@"Fotos" photos:pics photos2:nil] autorelease];

        }

}
```

4.1.5 Envío de favoritos por Bluetooth

Otro aspecto importante de la implementación es el envío de favoritos mediante Bluetooth. Para llevar a cabo esta tarea, se ha utilizado el Framework *Gamekit* [35] proporcionado por el SDK del iPhone a partir de la versión 3.0. Algunos aspectos importantes a tener en cuenta a la hora de implementar un desarrollo de conectividad bluetooth en iPhone son:

- La clase que se encargue de dicha tarea debe implementar los protocolos *GKSessionDelegate* y *GKPeerPickerControllerDelegate*.

- La implementación está basada en sesiones mediante objetos de la clase GKSession, a la cual se conectan los diferentes dispositivos (*peers*) que se deseen.

- El estandar IEEE 802.15 según el cuál se rige el bluetooth, permite conectar en una misma sesión de 5 a 7 dispositivos. A partir de entonces, la conectividad se puede volver inestable.

A continuación se verán algunas partes importantes de dicha implementación.

El siguiente método llamado *sendByBluetooth* se ejecuta cada vez que se pulsa el botón central de la barra superior en la vista de favoritos y es el encargado de mostrar una ventana indicando la búsqueda de dispositivos bluetooth cercanos ejecutando el programa (ver Figura 46). Una vez encontrados, permite al usuario seleccionar el dispositivo con el que se desee conectar.

```
-(void)sendByBluetooth{
        .........
        picker = [[GKPeerPickerController alloc] init];
        picker.delegate = self;
        picker.connectionTypesMask = GKPeerPickerConnectionTypeNearby;
        [picker show];
}
```

Figura 46: Búsqueda de dispositivos Bluetooth

En el momento en que el usuario A intenta conectar con B, pulsando el nombre que ve en la pantalla anterior, se ejecuta en B el siguiente método, aceptando la conexión de forma inmediata.

```
- (void)session:(GKSession*) session didReceiveConnectionRequestFromPeer:(NSString*) peerID {

        [currentSession acceptConnectionFromPeer:peerID error:nil];

}
```

Posteriormente, tanto en A como en B se ejecutan los dos siguientes métodos. En el primero simplemente se muestra una alerta de conexión con el usuario correspondiente. Dicho método se ejecuta de nuevo en la desconexión, liberando la sesión correspondiente.

```
- (void)session:(GKSession *)session peer:(NSString *)peerID didChangeState:(GKPeerConnectionState)
  state {

    switch (state)
    {
        .........
        case GKPeerStateConnected:

                                alert=[[UIAlertView alloc] initWithTitle:@"Conectado" message:[NSString
                                stringWithFormat:@"Acaba de conectarse con: ...];
                                ...
                                 break;

        case GKPeerStateDisconnected:

                                _alert=[[UIAlertView alloc] initWithTitle:@"Desconectado" message:[NSString
                                stringWithFormat:@"Acaba de desconectarse de: ...];
                                ....
                                break;
    }
}
```

El segundo de ellos asigna la sesión al dispositivo correspondiente para que ambos compartan la misma.

```
- (void)peerPickerController:(GKPeerPickerController *)picker didConnectPeer:(NSString *)peerID
                toSession:(GKSession *) session {

        self.currentSession = session;
        session.delegate = self;
        [session setDataReceiveHandler:self withContext:nil];
        picker.delegate = nil;

        .....
}
```

Una vez se tienen ambos dispositivos conectados, el siguiente paso es enviar la información, esto es, los lugares favoritos que se tengan almacenados. Tras diversas pruebas, se llegó a la conclusión de que el número máximo de favoritos a enviar para una recepción correcta y fiable por parte del otro terminal era de cuatro. El siguiente método muestra la parte del código encargada de dicha tarea. Varias consideraciones:

1. La variable *selectedRows* indica las filas correspondientes a los lugares a enviar (máximo cuatro).

2. Los distintos lugares se añaden al array *bluetooth_places* el cual se encapsula posteriormente en un objeto de la clase *NSData* para su correcta transmisión.

3. El método encargado de efectuar la transmisión (*sendDataToAllPeers*) devuelve un valor booleano que se utiliza para saber si el envío se ha realizado correctamente.

```
NSMutableArray *bluetooth_places=[NSMutableArray new];

for (int i=0 ; i<[selectedRows count]; i++)
        [bluetooth_places addObject:[a objectAtIndex:[[selectedRows objectAtIndex:i] indexAtPosition:1]]];

NSData *data = [NSKeyedArchiver archivedDataWithRootObject:bluetooth_places];

BOOL correctSend=[self.currentSession sendDataToAllPeers:data withDataMode:GKSendDataReliable
error:nil];
```

El método encargado de recibir la información se puede ver a continuación.

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer inSession:(GKSession *)session context:(void *)context {

        NSArray *a=[NSKeyedUnarchiver unarchiveObjectWithData:data];

         for (int i=0; i<[a count]; i++) {

                        TouristicPlace *t=[[TouristicPlace alloc] initWithName:....];
                        .....
                        for (int i=0; i<[[favPlaces favouritePlaces] count]; i++) {
                                ....
                                found=YES;
                        }

                         if (!found) {
                                .....
                                 NSUserDefaults *prefs=[NSUserDefaults standardUserDefaults];
                                 [prefs removeObjectForKey:@"Favoritos"];
                                 [prefs setObject:[sf storeFavourites] forKey:@"Favoritos"];
                                 [prefs synchronize];
                         }
        }

        //Update badge and alert user

         }
}
```

El método realiza seis acciones principales:

1. Reconstruir el array de lugares desencapsulando el objeto *NSData* recibido.

2. Se construyen los distintos objetos *TouristicPlace* a partir del array anterior.

3. Se comprueba si alguno de los favoritos recibidos se tenía ya guardado como tal anteriormente.

4. Si no se tiene, se agrega, se guarda en *NSUserDefaults* para que se almacene el favorito y aparezca en las distintas sesiones de uso de la aplicación.

5. Actualizar la lista de favoritos.

6. Actualizar el *badge* (icono rojo junto a la pestaña favoritos que indica el número de éstos que se tienen guardados).

4.1.6 Caracteres especiales HTML

El hecho de haber tenido que parsear el código fuente de distintas páginas webs para conseguir la distinta información multimedia de los diferentes lugares, ha provocado que se tuviera que tratar con ciertos aspectos de HTTP y su codificación de caracteres. La mayoría de los navegadores usan la codificación ISO-8859-1 [32]. En la Tabla 23 se pueden ver algunos ejemplos de dicha codificación.

Símbolo	Nombre de la entidad	Número de la entidad
-	–	–
¿	¿	¿
&	&	&
á	á	á
ç	ç	ç

Tabla 23: Codificación ISO-8859-1

Se puede ver como para cada símbolo (o entidad como se conoce en HTML) existen dos tipos de codificaciones, basadas en el nombre o en el número de la entidad. Dependiendo del navegador, se utilizará una u otra. Parte del método implementado para tratar con dicha codificación se puede ver a continuación.

```
-(NSString *)unscape:(NSString *)scapedString{

        NSString *unescapedString=[scapedString mutableCopy];

        unescapedString=[unescapedString stringByReplacingOccurrencesOfString:@"&#8230;"
withString:@"..."];
        unescapedString=[unescapedString stringByReplacingOccurrencesOfString:@"&#8217;"
withString:@"'"];
        unescapedString=[unescapedString stringByReplacingOccurrencesOfString:@"&#8216;"
withString:@"'"];
        unescapedString=[unescapedString stringByReplacingOccurrencesOfString:@"&iacute;"
withString:@"í"];

.........
        unescapedString=[unescapedString stringByReplacingOccurrencesOfString:@"%3D"
withString:@"="];
        unescapedString=[unescapedString stringByReplacingOccurrencesOfString:@"%2B"
withString:@"+"];

        return unescapedString;
}
```

4.1.7 Google Mobilizer

Aunque la mayoría de las páginas webs disponen hoy en día de una versión para dispositivos móviles, todavía existen muchas páginas que resulta difícil leer en un dispositivo con una pantalla de 3.2 pulgadas. Para ello existen distintos servicios que intentar mejorar la experiencia de lectura en las pequeñas pantallas de los dispositivos móviles. Los dos más famosos son:

1. Google Mobilizer [33]
2. Instapaper Mobilizer [34]

En el proyecto se ha utilizado el primero de ellos, utilizando la siguiente URL proporcionada por Google: *http://www.google.com/gwt/x?u=.* Así, si se desea ver la página de la Universidad Carlos III de Madrid mediante Google Mobilizer, la URL a introducir sería: *http://www.google.com/gwt/x?u=www.uc3m.es.*

Se puede ver como convertir una página web para mejorar su lectura en pantallas pequeñas es tan sencillo como anteponer a la URL la sentencia *http://www.google.com/gwt/x?u=.* En la Figura 47 se muestra el resultado y la diferencia que supone utilizar este método cuando se diseña software para dispositivos móviles.

Figura 47: Google Mobilizer

4.2 Resultados del plan de pruebas

El resultado del plan de pruebas ha sido satisfactorio superándose todas y cada una de ellas.

Destacar que la prueba PR-13, donde se trata de asegurar que la interfaz de usuario sea acorde a los estándares preestablecidos por Apple, es ciertamente subjetiva. Aún así, en el anexo A se puede ver la simpleza y sencillez tanto de la interfaz como de los controles creados para la aplicación, por lo que dicha prueba se puede considerar superada.
Notar que la prueba PR-10 relacionada con Twitter, pese a que si se realizara en el momento en que se están escribiendo estas líneas no se superaría, como se mencionó previamente, dicha autenticación funcionó perfectamente hasta el 31 de agosto de 2010, donde Twitter puso en marcha el protocolo *OAuth* en su API.

Capítulo 5: Gestión del proyecto

En este capítulo se presentan los diferentes estados del proyecto en lo relativo a su gestión mediante la comparación de la planificación inicial y la real (las cuales difieren considerablemente como se verá a posteriori), así como el diferente hardware y software utilizado para la realización del mismo. Por último, se detalla un análisis económico exhaustivo donde se estudian dos posibilidades de ingreso de capital a través de la aplicación desarrollada. Destacar que se ha adaptado la plantilla de presupuestos existentes para Proyectos Fin de Carrera para ajustarlo a las características del documento, tratándose de un presupuesto totalmente equivalente.

- A través de la venta de la aplicación por los medios habituales.

- A través de la nueva plataforma de publicidad de Apple, llamada iAD, ofreciendo la aplicación de forma gratuita.

5.1 Planificación del proyecto

En este apartado se detallarán las tareas más importantes que han compuesto el proyecto junto con el esfuerzo en días necesario para llevar a cabo cada una de ellas. Esto se dispondrá gráficamente en forma de diagrama de Gantt. Se verá una comparación de la planificación ideal o inicial con la que posteriormente fue la real y verdadera.

5.1.1 Planificación inicial

A continuación se presentarán las diferentes fases o tareas por las que inicialmente se previó que pasara el proyecto. Como se verá posteriormente en el desarrollo real, esta planificación difirió mucho con respecto a la verdadera. En un principió, las tareas inicialmente planteadas fueron:

- **Análisis**

  ✦***Análisis de las necesidades***: Necesidades y requisitos básicos del software a desarrollar.

  ✦***Estudio de las plataformas móviles***: En esta tarea se realizó una investigación sobre las plataformas móviles actuales, el estado del arte de las mismas, y la conveniencia de elegir una u otra para la aplicación en cuestión a desarrollar.

✦***Estudio de las herramientas de programación***: Aquí se engloba el aprendizaje tanto del software específico para desarrollo de aplicaciones para iPhone como del lenguaje de programación específico de la plataforma así como del SDK (*Software Development Kit*) del iPhone.

✦***Obtención de lugares cercanos***: Estudio de servicios web y herramientas necesarias para obtener lugares cercanos a la posición actual. Estudio del API de Google Maps. Posibilidad de utilizar KML (*Keyhole Markup Language*) o JSON (*JavaScript Object Notation*).

✦***Identificación de valores añadidos***: Aquí se analizó las características más necesarias y relevantes de cara al sector turístico que era necesario disponer en cada lugar (fotos, vídeos, opiniones...).

✦***Análisis de la realización de rutas***: Aquí se analizó las diferentes necesidades, medios disponibles y servicios web necesarios para realizar rutas entre diferentes lugares.

- **Diseño**

✦***Diseño de la interfaz***: Esta tarea consistió en elaborar las líneas básicas que tendría la interfaz y los distintos elementos con los que interecciona ría el usuario. Notar la especial importancia que tiene esta tarea en la plataforma iPhone, donde se requieren unos patrones y líneas de diseño muy concretos para conseguir una experiencia de usuario (UX) acorde a la plataforma.

✦***Diseño del funcionamiento*****:** Etapa en la que se realiza el diseño detallado de las distintas clases que forman el software.

- **Desarrollo**

✦***Implementación básica***: En esta tarea se realizó la programación relacionada con la localización del usuario a través del API específica, la obtención de los lugares cercanos a través del parseo del fichero JSON obtenido de las APIS de Google Maps y la presentación en pantalla al usuario tanto en modo lista como en modo mapa.

✦I***mplementación de los valores añadidos:*** implementación del código necesario para conseguir fotos, vídeos y opiniones de los diferentes lugares de interés.

✦**Integración de servicios básicos**: Esta tarea supone aprovechar las capacidades específicas del teléfono móvil para poder realizar acciones específicas sobre los

lugares, esto es, llamar por teléfono, recomendar a un amigo por e-mail o agregar a la agenda el contacto para una consulta posterior.

✦**Rutas**: En esta tarea se implementará todo lo necesaria para poder realizar rutas entre los diferentes lugares. Dichas rutas se consiguen a través del parseo del fichero JSON del API de *CloudMade*. Más adelante se explicará en más detalle.

- **Pruebas:** Tarea estándar donde se intenta optimizar el código mediante la depuración y una serie de pruebas que se comentarán más adelante.

- **Documentación:** Realización del presente documento.

A continuación se puede ver el diagrama de Gantt ideal que se planificó inicialmente.

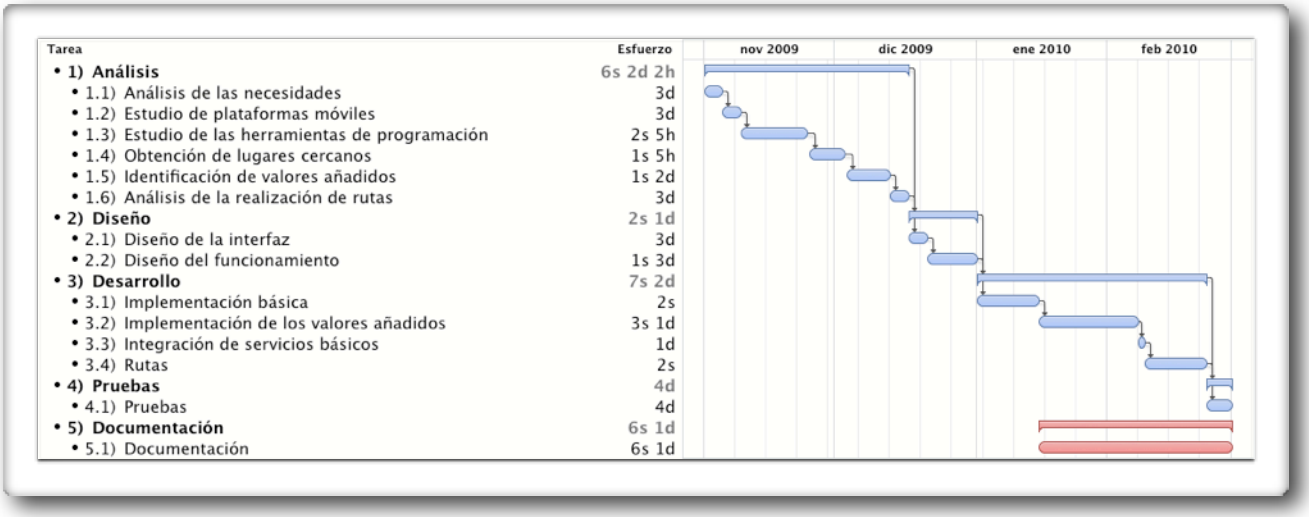

Figura 48: Planificación inicial

5.1.2 Planificación real

La planificación inicial se cumplió de forma muy satisfactoria hasta el momento en que, debido a temas laborales, se disminuyó radicalmente las horas dedicadas al proyecto, lo que alargó de forma muy importante la duración del mismo.

Cabe destacar que, debido al uso del ciclo de vida en cascada con realimentación, en las distintas revisiones de la fase de análisis se decidió añadir las dos siguientes tareas, las cuales suponen un gran potencial de cara al usuario y a su experiencia final al usar la aplicación

- **Redes sociales**: Hoy en día, es casi un requisito dotar a la aplicación de capacidades 2.0 y de redes sociales, y esta tarea pretende justo eso, permitir al usuario recomendar a sus amigos cualquier sitio tanto en Facebook como en Twitter. La implementación se realizó a través de las API proporcionadas por cada uno de estos dos servicios.

- **Bluetooth**: Implementación de la posibilidad de intercambiar lugares favoritos a través de la conectividad Bluetooth de que disponga el terminal (en caso de que tenga).

La Figura 49 presenta el diagrama de Gantt que refleja el desarrollo real de las tareas.

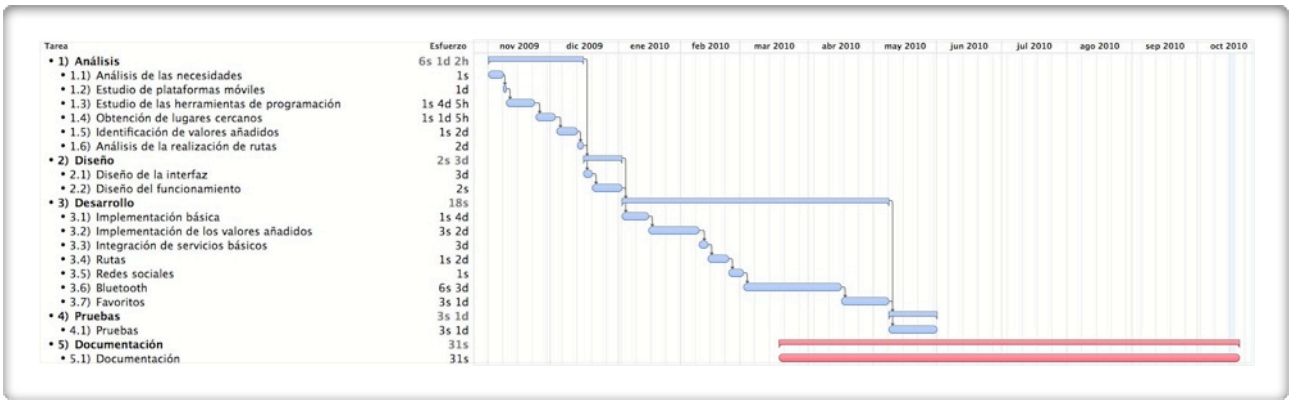

Figura 49: Planificación real

Algunas tareas requirieron más esfuerzo del planteado, como por ejemplo el análisis de las necesidades o el diseño del funcionamiento. Otras sin embargo requirieron menos esfuerzo del esperado, como por ejemplo la inclusión de las rutas.

Se puede apreciar como a partir de marzo 2010, cuando la dedicación al proyecto disminuyó de forma muy considerable, las tareas que se realizaron fueron la integración de funcionalidad bluetooth a la aplicación, las pruebas/depuración y la presente documentación. También se realizaron frecuentes cambios de detalles relacionados con la interfaz/optimización del programa que no han sido detallados en el Gantt.

5.2 Medios técnicos empleados

Aquí se presentarán las diferentes herramientas tanto hardware como software que han sido utilizadas durante toda la realización del proyecto. Aunque algunas de los programas citados a continuación no son estrictamente imprescindibles para la realización del proyecto, han sido de gran ayuda y se ha decidido incluirlos también en el análisis económico.

5.2.1 Hardware

En la Tabla 24 se pueden ver los distintos dispositivos hardware que se han utilizado a lo largo de la realización del proyecto.

Dispositivos Mac OS X	Portátil Apple MacBook, procesador Intel Core 2 Duo 2 GHz, 2 GB de RAM	Apple iMac 27", procesador Intel Core 2 Duo 3.06 GHz, 4 GB de RAM
Dispositivos iOS	iPhone 2G/iPhone 4	Dos iPod Touch 3G

Tabla 24: Dispositivos hardware utilizados

5.2.2 Software

En la Tabla 25 se pueden ver todo el software utilizado en la vida del presento Proyecto Fin de carrera.

Tipo	Nombre	Página Web
Sistemas operativo	Apple Mac OS X 10.6 Snow Leopard	http://www.apple.com/es/macosx/
Entorno de programación	Xcode 3.2.1	http://developer.apple.com/technologies/tools/
Entorno de diseño de interfaces:	Interface Builder 3.2.1	http://developer.apple.com/technologies/tools/
Gestión de memoria	Instruments 2.0.1	http://developer.apple.com/technologies/tools/
Gestión de tareas:	Things	http://culturedcode.com/
Herramienta de control de versiones	Git, GitX	http://git-scm.com/ http://gitx.frim.nl/
Tratamiento de imágenes:	Pixelmator 1.5.1	http://www.pixelmator.com/

Tipo	Nombre	Página Web
Gestión del proyecto:	OmniPlan 1.6.3	http://www.omnigroup.com/products/omniplan/)
Gestión de capturas de imagen	LittleSnapper 1.6	http://www.realmacsoftware.com/littlesnapper/)
Gestión de capturas del simulador del iPhone	iPhone ScreenTaker	http://fabian-kreiser.com/
Diagramas	OmniGraffle 5.2.1	http://www.omnigroup.com/products/omnigraffle/
Procesador de textos	Apple Pages 09	http://www.apple.com/es/iwork/
Presentaciones	Apple Keynote 09	http://www.apple.com/es/iwork/

Tabla 25: Herramientas de software utilizadas

5.3 Análisis económico

En esta sección se procede a detallar cada uno de los costes que suponen cada una de las partes que componen el proyecto, esto es, RRHH, hardware y software. Destacar que se comparará el presupuesto inicial con el presupuesto final para calcular la desviación en que se ha incurrido.

Así mismo y como se comentó anteriormente, se hará un estudio de la estrategia de venta más acorde, verticalizando la solución en base a las dos opciones mencionadas al principio del capítulo. De cualquier modo, se ha descartado la venta del producto a un cliente externo, para conseguir los ingresos por los medios habituales de distribución digital.

5.3.1 Metodología de estimación de costes

Para realizar una estimación de costes correcta y adecuada al caso de un desarrollo software como es el que se ha desarrollado, se intentará dividir en cuatro grandes grupos: personal, hardware, software y costes indirectos. La adición de estos cuatro

costes suponen una aproximación muy buena al verdadero coste del proyecto. A continuación se analizarán detalladamente una por una.

- **Personal:** El coste de personal se basará en los honorarios del Ingeniero de Telecomunicación obtenidos a través de la página del Colegio Oficial de Ingenieros de Telecomunicación (COIT).

- **Hardware:** En el caso del hardware, se aplicará el prorrateo correspondiente de cada elemento hardware según el tiempo que se haya utilizado en el proyecto. Nótese que no se incluirá en el cálculo todo el hardware utilizado y mencionado antes, sino aquel estrictamente necesario para la realización del proyecto.

- **Software:** En este apartado se estimará el coste a través del precio de las licencias de cada uno de los programas utilizados.

- **Costes indirectos:** Aquí se tendrán en cuenta, entre otros, los siguientes gastos:

    - Gastos de luz.
    - Gastos de redes de comunicaciones

5.3.2 Análisis de costes estimados

En este punto se analizará el coste del proyecto orientativo que se pensó antes de iniciarse el proyecto. Como se verá en el presupuesto real, las diferencias entre ambos son muy grandes debido a los motivos explicados en la planificación real.

5.3.2.1 Estimación del coste de personal

Los costes de personal o recursos humanos (RRHH) se componen únicamente de los honorarios del Ingeniero de Telecomunicación encargado del desarrollo del proyecto. Los honorarios correspondientes a un Ingeniero de Telecomunicación se pueden ver en la página web del Colegio Oficial de Ingenieros de Telecomunicación[21]. En un primer momento, se planificó el proyecto con una carga horaria de 8 horas diarias durante 4 meses (supóngase que un mes equivale a 20 días laborables).

Concepto	Horas	Honorarios	Coste RRHH
Ingeniero de Telecomunicación	640 horas	47 €/hora	30.080 €

Tabla 26: Coste de personal estimado

[21] http://www.coit.es

5.3.2.2 Estimación del coste del Hardware

A continuación se especificará un listado del hardware utilizado durante la realización del proyecto junto con su precio para poder estimar el coste que supone el hardware en el mismo.

Notar que, como se comentó previamente, sólo se procederá a facturar el hardware que ha sido estrictamente necesario para la realización del proyecto, y no aquel que se ha utilizado a modo de extra, como por ejemplo el iMac 27" con el que se desarrolló parte del proyecto en el trabajo.

Concepto	Unidades	Precio unitario	Vida útil estimada	Tiempo de uso	Coste para el proyecto
Portátil Apple MacBook	1	1099 €	48 meses	4 meses	91,58 €
iPod Touch 3G 32 GB	2	269 €	48 meses	2 meses	22,41 €
					113,99 €

Tabla 27: Coste de hardware estimado

Destacar que todos los precios tienen incluido el 18% de IVA.

5.3.2.3 Estimación del coste del Software

A continuación se hará un análisis económico del diferente software utilizado. Destacar el hecho de la donación de 10 € en las distintas herramientas de software libre que se han utilizado para ayudar y fomentar el desarrollo continuo de estas aplicaciones por parte de los creadores.

Por último mencionar que para poder subir una aplicación desarrollada para iPhone/iPad a la tienda de aplicaciones del dispositivo (se explicará más sobre esto más adelante), es necesario pagar la licencia *iPhone Developer Program* de $99.

Concepto	Coste licencia	Vida útil estimada	Tiempo de uso	Coste para el proyecto
Mac Box Set (Apple Mac OS X 10.6 Snow Leopard + iWork '09)	169 €	48 meses	4 meses	14,08 €
Omniplan	121,05 €	12 meses	1 mes	10,08 €
OmniGraffle	80,68 €	12 meses	2 meses	13,44 €
LittleSnapper	34,62 €	12 meses	2 meses	5,77 €
Things	49,95 €	12 meses	4 meses	16,65 €
Pixelmator	45,62 €	12 meses	2 meses	7,6 €
Xcode, Interface Builder, Instruments	0 €	-	4 meses	0 €
Git	10 €	12 meses	4 meses	3,33 €
GitX	10 €	12 meses	4 meses	3,33 €
iPhone ScreenTaker	10 €	12 meses	4 meses	3,33 €
iPhone Developer Program	79,92 €	-	-	79,92 €
				157,53 €

Tabla 28: Coste de software estimado

5.3.2.4 Estimación de los costes indirectos

Como se mencionó previamente, es complicado establecer una lista de todos los costes indirectos aplicables en el proyecto. Se intentará estimar mediante la inclusión de la conexión a Internet y el gasto de luz en que se ha incurrido.

Para la conexión a Internet, supóngase que la tarificación mensual por parte del proveedor de servicios de Internet correspondiente es de 42 €/mes. En cuanto a la luz, se aplicará un gasto mensual de 50 €/mes. Si se tiene en cuenta la duración estimada inicial del proyecto (4 meses), se tendría la siguiente tabla de costes indirectos.

Concepto	Precio mensual	Tiempo de uso	Coste para el proyecto
Conexión a Internet	42 €	4 meses	168 €
Luz	50 €	4 meses	200 €
			368 €

Tabla 29: Costes indirectos estimados

5.3.2.5 Estimación de los costes totales

En la siguiente tabla se muestra el coste total en que se ha incurrido con la realización del proyecto, que no es más que la suma de los costes calculadores previamente.

Concepto	Coste
Coste de personal	30.080 €
Coste de hardware	113,99 €
Coste de software	157,53 €
Costes indirectos	368 €
	30.719,52 €

Tabla 30: Costes totales estimados

Por tanto, el coste total que se espera tener en la realización del proyecto es de **30.719,52 €**. En el siguiente apartado se verá el error en el que se ha incurrido con el coste real.

5.3.3 Análisis de costes reales

En este apartado se va a calcular el coste real que se ha tenido una vez se ha realizado el proyecto. La principal diferencia con respecto al coste total inicial calculado en el apartado anterior, es la diferencia sustancial de horas.

Durante los primeros cuatro meses del proyecto (noviembre 2009 - febrero 2010) se dispuso de todo el tiempo para la realización del proyecto, aunque en vez de una dedicación de 8 horas inicial, al final se terminó dedicando una media 5 horas útiles por día al mismo. Desde marzo 2010 hasta la finalización del proyecto (octubre de 2010), la

dedicación fue muy poca e irregular debido a motivos laborales. Supóngase que se ha dedicado, en media, una única hora diaria al proyecto.

Dedicación	Horas diarias dedicadas	Meses	Horas totales
Tiempo completo	5	4	400
Tiempo parcial	1	7,66	153,2
			553,2

Tabla 31: Horas a tiempo completo y parcial

Notar que los 7.66 meses tienen en cuenta la duración del proyecto hasta el 20 de octubre de 2010.

Aunque las horas totales de dedicación al proyecto son menores que las 640 horas con las que se contó en la planificación inicial, con la consecuente disminución del coste de recursos humanos, el precio del hardware, software y costes indirectos será mayor debido al mayor número de meses que se ha utilizado estos bienes. A continuación se calculan las diferencias en cada uno de los apartados.

5.3.3.1 Coste real de personal

En cuanto al personal, la disminución del tiempo dedicado al proyecto implica una disminución proporcional en el coste de RRHH como muestra la siguiente tabla.

Concepto	Horas	Honorarios	Coste RRHH inicial	Coste RRHH real	Diferencia
Ingeniero de Telecomunicación	553,2 horas	47 €/hora	30.080 €	26.000,4 €	4.079,6 € (-)

Tabla 32: Coste de personal real

5.3.3.2 Coste real del Hardware

En cuanto al hardware, el software y los costes indirectos, existen cambios aplicables al prorrateo como consecuencia del aumento de meses en la duración del proyecto.

Concepto	Vida útil estimada	Tiempo de uso	Coste HW inicial	Coste HW real	Diferencia
Portátil Apple MacBook	48 meses	11,66 meses	91,58 €	266,96 €	€ 175,38 (+)
iPod Touch 3G 32 GB	48 meses	2 meses	22,41 €	22,41 €	€ 0
			113,99 €	289,37 €	175,38 € (+)

Tabla 33: Coste de hardware real

5.3.3.3 Coste real del Software

Concepto	Vida útil estimada	Tiempo de uso	Coste SW inicial	Coste SW real	Diferencia
Mac Box Set (Apple Mac OS X 10.6 Snow Leopard + iWork '09)	48 meses	11,66 meses	14,08 €	40,83 €	26,75 € (+)
Omniplan	12 meses	1 mes	10,08 €	10,08 €	0 €
OmniGraffle	12 meses	3 meses	13,44 €	20,17 €	6,73 € (+)
LittleSnapper	12 meses	8 meses	5,77 €	23,08 €	17,31 € (+)
Things	12 meses	11,6 meses	16,65 €	48,28 €	31,63 € (+)
Pixelmator	12 meses	3 meses	7,6 €	11,4 €	3,8 € (+)
Xcode, Interface Builder, Instruments	-	7 meses	0 €	0 €	0 €
Git	12 meses	7 meses	3,33 €	5,82 €	2,49 € (+)
GitX	12 meses	7 meses	3,33 €	5,82 €	2.49 € (+)

Concepto	Vida útil estimada	Tiempo de uso	Coste SW inicial	Coste SW real	Diferencia
iPhone ScreenTaker	12 meses	11,66 meses	3,33 €	9,7 €	6,37 € (+)
iPhone Developer Program	-	-	79,92 €	79,92 €	€ 0
			157,53 €	255,1 €	97,57 € (+)

Tabla 34: Coste de software real

5.3.3.4 Costes indirectos reales

Concepto	Precio mensual	Coste indirecto inicial	Coste indirecto real	Diferencia
Conexión a Internet	€ 42	168 €	489,72 €	321,72 € (+)
Luz	€ 50	200 €	583 €	383 € (+)
		368 €	1072,72 €	704,72 € (+)

Tabla 35: Costes indirectos reales

5.3.3.5 Costes totales reales

Concepto	Coste total inicial	Coste total real	Diferencia
Coste de personal	30.080 €	26.000,4 €	4.079,6 € (-)
Coste de hardware	113,99 €	289,37 €	175,38 € (+)
Coste de software	157,53 €	255,1 €	97,57 € (+)
Costes indirectos	368 €	1072,72 €	704,72 € (+)
	30.719,52 €	**27.617,59** €	**3.101,93 € (-)**

Tabla 36: Costes totales reales

Por tanto, el coste final para el proyecto ha sido de **27.617,59 €** con una desviación de tan **3.101,93 €** con respecto a la planificación estimada inicial**.** Esto supone un 10.1 % de ahorro sobre el presupuesto inicial de 30.719,52 euros. Este dato, aunque nada despreciable, se puede considerar como una medida de que la planificación inicial fue una aproximación muy aceptable del resultado final real.

5.4 Análisis de la forma de venta de la aplicación

Como se comentó anteriormente, el objetivo del desarrollo de la aplicación es obtener un beneficio de la misma (como es lógico) pero no a través de la venta a un cliente externo, sino a través de la propia venta de la aplicación por el método habitual de distribución digital, esto es, la tienda de aplicaciones para iPhone (también conocida como App Store). Se explica a continuación muy brevemente en qué consiste esta plataforma.

La App Store surge el 10 de julio de 2008, como una plataforma de descarga digital para las aplicaciones desarrolladas para iPhone y iPod Touch (ahora también para iPad). Es una forma de intentar homogeneizar la venta digital de aplicaciones móviles y facilitar al desarrollador los gastos de distribución de su aplicación. Para poder subir una aplicación a esta tienda de aplicaciones, es necesario abonar el precio de la licencia "*iPhone Developer Program*" de $99 anuales. La ganancia que tiene el desarrollador es el 70% de todo el dinero generado por la aplicación, mientras que Apple se quedará con el 30% restante. Igualmente se puede ofrecer la aplicación de forma gratuita. En la Figura 50 se puede ver la App Store desde tres canales diferentes (PC/mac, iPhone/iPod Touch e iPad).

Figura 50: Multicanalidad de la App Store

Una vez se tiene la panorámica necesaria para entender el modelo de negocio que establece Apple en sus dispositivos, se procede a analizar las dos siguientes opciones:

1. Vender la aplicación a un precio específico para, según el número de ventas esperadas, conseguir el tanto por ciento de beneficio deseado.

2. Utilizar el nuevo sistema publicitario ofrecido por Apple para la creación de anuncios integrados en la propia aplicación llamado iAd, de tal forma que se pueda ofrecer la aplicación de forma gratuita.

Se analizarán ambas dos desde un punto de vista de mayor retorno de la inversión o ROI[22] y se tratará de justificar la elección tomada.

5.4.1 Venta ordinaria no gratuita

Supóngase que se desea obtener un ROI del 30%, esto implica que el importe económico que se quiere tener es de 36.000 €. La siguiente gráfica muestra el número de aplicaciones que se deberían vender para conseguir tal cantidad según el precio de venta correspondiente (recuérdese que el desarrollador recibe el 70%, mientras que el 30% restante es para Apple).

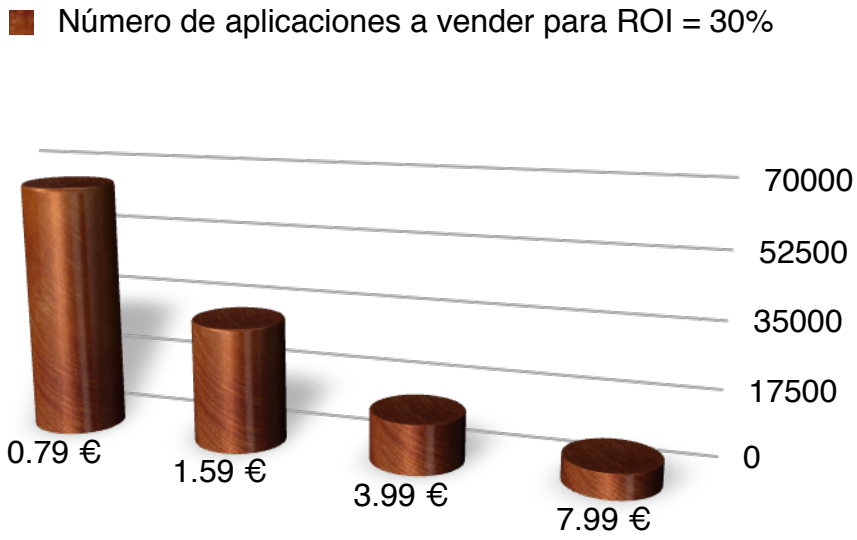

Figura 51: Número de aplicaciones a vender para ROI = 30%

[22] ROI - El retorno de la inversión (*Return Of Investment)* es un porcentaje que se calcula como el cociente entre los beneficios y los costes iniciales. Par más información ver http://www.stickyminds.com/sitewide.asp?ObjectId=6525&Function=edetail&ObjectType=ART

Precio	Número de unidades a vender	Descargas diarias	Años para ROI = 30%
€ 0,79	65.100	30	5,94
€ 1,59	32.346	30	2,95
€ 3,99	12.890	30	1,17
€ 7,99	6.437	30	0,58

Tabla 37: Años para ROI = 30% variando precio

Existen en la actualidad alrededor de 100 millones de dispositivos que corren iOS, y por tanto susceptibles de descargar la aplicación. De ahí la suposición de 30 descargas diarias para la misma. Si se elige vender la aplicación a 3.99 €, lo que supone un precio ajustado al tiempo dedicado y la tecnología empleada, en aproximadamente un año se habrá alcanzado el beneficio buscado del 30%, es decir, unas ganancias de 8382 €.

Debido al gran carácter adquisitivo del mercado al que va dirigida la aplicación, el número de descargas diarias puede ser considerablemente mayor al estimado y con una gran variabilidad. En las dos siguientes tablas se analiza qué ocurre con la variación de este ratio, tanto con respecto a la ROI obtenida como con los días necesarios para alcanzar la ROI fija del 30% respectivamente.

Precio	Descargas diarias	Número de días	ROI
€ 3,99	25	430	8,7%
€ 3,99	30	430	30%
€ 3,99	60	430	160,9%
€ 3,99	90	430	291,3%

Tabla 38: ROI variando las descargas diarias

Como se puede ver en la Tabla 38, y como era lógico, la ROI aumenta exponencialmente según el aumento lineal de las descargas diarias.

Precio	Número de unidades a vender	Descargas diarias	Años para ROI = 30%
€ 3,99	12.890	25	1,41
€ 3,99	12.890	30	1,17
€ 3,99	12.890	60	0,58
€ 3,99	12.890	90	0,39

Tabla 39: Años para ROI = 30% variando descargas diarias

5.4.2 Venta a través del sistema publicitario iAd

La otra posibilidad que se estudiará es la posible utilización de la plataforma publicitaria iAd, presentada por Apple el 8 de abril de 2010, y que entró en funcionamiento el 1 de julio de 2010. En los dos siguientes apartados, se explicará qué es y qué supone este nuevo sistema publicitario, así como se realizará un análisis económico de los beneficios esperados en el caso de que se adopte esta opción.

5.4.2.1 Definición de iAd

iAd es el nuevo nombre que recibe la publicidad móvil dentro de las aplicaciones nativas para iPhone. Está basada en contenido y, de alguna forma, los anuncios estarán relacionados con la temática de la aplicación en la cual se ejecuta el anuncio.

Otro aspecto interesante de iAd es la utilización del modelo CPM (*Cost Per Mille* = *Cost Per Thousand views*) + CPC (*Cost Per Click*). El anunciante paga una cantidad concreta de dinero por cada mil veces que su anuncio se muestre en la aplicación (impresión) y otra cantidad mucho mayor cada vez que se haga click en el anuncio.

Los diferentes actores en este nuevo negocio son:

- **Anunciantes**: pagan para tener sus anuncios en las aplicaciones para iPhone.
- **El desarrollador de la aplicación**: obtiene el 60% de los beneficios generados por el anuncio.
- **Apple**: se queda con el 40% restante.
- **El usuario de la aplicación**: se beneficia del abaratamiento de las aplicaciones

Por tanto, iAd supone un medio para que los desarrolladores puedan ofrecer sus aplicaciones de forma gratuita obteniendo los ingresos a través de la publicidad, beneficiándose los usuarios de ello.

Destacar que dicho modelo no sólo beneficia a las aplicaciones que se venden a un precio concreto, sino que de igual manera las aplicaciones que antes eran gratuitas pueden ahora optar por añadir anuncios para conseguir nuevos ingresos. Si se tiene en cuenta que el 25% de las aplicaciones en la App Store son gratuitas, esto supone 50.000 aplicaciones que ahora tienen una vía para generar ingresos antes inexistentes.

5.4.2.2 Análisis económico de iAd

Supóngase que existen 40 millones de iPhones activos en la actualidad. El número de aplicaciones descargadas hasta la fecha ronda los 4 billones. Si se supone que sólo el 10% de las aplicaciones descargadas por el usuario son susceptibles de usarse en el día a día, eso supone 400 millones de aplicaciones. Si se divide este número por los 40 millones de dispositivos y se aplica el 25% correspondiente a las aplicaciones gratuitas de la App Store implica que un usuario cualquiera dispone de 2-3 aplicaciones susceptibles de mostrar iAds. Esto implicaría aproximadamente de 5 a 10 impresiones de iAds en un sólo usuario y de 200 a 400 millones de impresiones por día en toda la comunidad de iPhones. Si se extrapola este valor a un año completo, se tiene un número de impresiones de 73 a 145 billones por año. Teniendo en cuenta que el modelo CPM premium rondaría de media los $15 por cada 1000 impresiones, las ganancias son ingentes para Apple.

Ahora se verá si de verdad supone la misma cantidad de ganancia para un desarrollador en concreto, aplicando el modelo CPM a la aplicación desarrollada. Al igual que antes, se quiere llegar a tener un 30% de ROI, es decir, conseguir 33800€. Se tomará un valor muy conservador para el modelo CPC, donde sólo el 1% de los usuarios que ve impreso el anuncio, hace click en él. Por tanto, teniendo en cuenta que el desarrollador obtiene el 60% de los beneficios, la fórmula será:

$$36.000 € = X * Y * (Z * 0.99 + 0.01 * 1.5€) * 0.6 \quad (I)$$

donde X es el número de impresiones al día, Y el número de días y Z el dinero en euros que paga el anunciante por cada 1000 impresiones.

Para estimar las impresiones/día, hay que tener en cuenta que ahora, al tratarse de una aplicación gratuita, las descargas de la aplicación se incrementarán muchísimo. Antes se supuso un flujo continuo de 30 descargas/día. En el nuevo modelo se va a suponer un incremento del 100%, teniendo 60 descargas diarias. El número de usuarios de iOS que, en media, tendrán la aplicación en su dispositivo será:

$$Nº \text{ de apps en media} = 60 * (Y + 1) / 2 \quad (II)$$

Para intentar ser realistas e intentar reflejar el hecho de que un usuario que acaba de descargar la aplicación supone más impresiones por pantalla en iAd que el usuario que se descargó la aplicación hace 3 meses y apenas la usa, se supondrá que sólo el 70% de los usuarios con la aplicación en su dispositivo aportan esas dos impresiones por día que se comentó anteriormente. Así, el número efectivo de impresiones en Y días será:

*X = Nº de apps en media * 2 impresiones/día * 0.7 = 60 * (Y + 1) * 0.7 (III)*

Por tanto, la fórmula que relaciona el número de días en que se alcanzará el ROI deseado y el precio que paga el modelo CPM + CPC por cada mil impresiones será:

*60 * (Y+1) * Y * (Z * 0.99 + 0.01 * 1.5€) * 0.6 * 0.7 = 36.000 (IV)*

Resolviendo dicha ecuación de segundo grado para varios valores de Z se tiene:

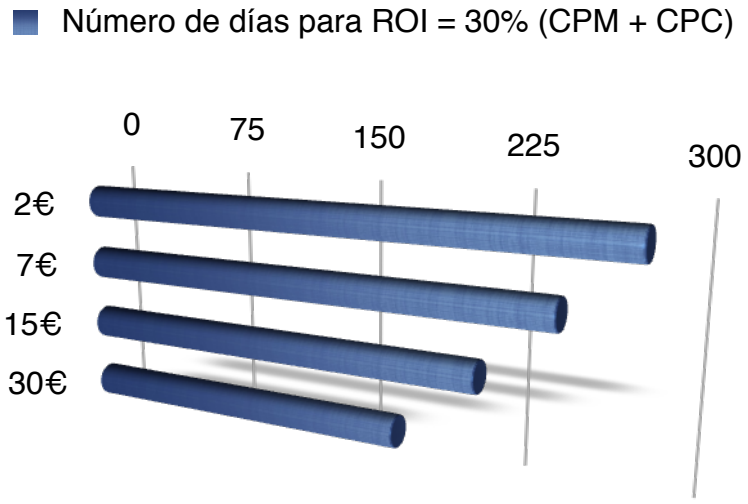

Figura 52: Número de días para ROI = 30% (CPM + CPC)

Precio por click (CPC)	Precio por 1000 impresiones (CPM)	Número de días para ROI = 30%
€ 1,5	€ 2	281
€ 1,5	€ 7	247
€ 1,5	€ 15	212
€ 1,5	€ 30	173

Tabla 40: Días para ROI = 30% variando el precio CPM (CPM + CPC)

La conclusión que se deriva de la Figura 52 y la Tabla 40 es que, cuando se aplican los sistemas CPM (coste por cada 1000 impresiones) y CPC (coste por click), el impacto que tiene CPM no es tanto como se esperaba (el número de días no desciende acorde a la subida de precio efectuada). Esto es debido a que el precio pagado al desarrollador cuando se hace click en el anuncio es mucho mayor que por la simple impresión. Resumiendo, CPC enmascara en gran medida a CPM.

Véase en el siguiente gráfico el caso en el que nadie hace click en el anuncio, o directamente los anunciantes sólo promueven un modelo CPM.

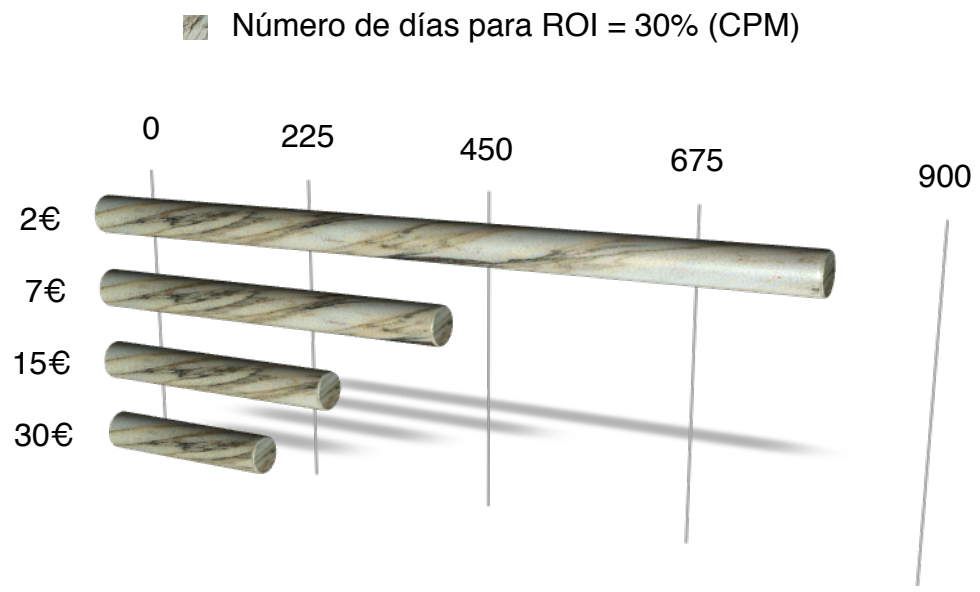

Figura 53: Número de días para ROI = 30% (CPM)

Precio por click (CPC)	Precio por 1000 impresiones (CPM)	Número de días para ROI = 30%
€ 1,5	€ 2	819
€ 1,5	€ 7	438
€ 1,5	€ 15	299
€ 1,5	€ 30	210

Tabla 41: Días para ROI = 30% variando precio (CPM)

Para conseguir un número de días parecido al análisis anterior de venta sin iAd (430 días), se debería tener un precio de 7 € por cada mil impresiones. El problema radica en que tal cantidad puede resultar muy difícil de conseguir para un desarrollador convencional a no ser que se consiga llegar a los puestos más altos en la App Store. Las empresas de publicidad pagarán más por estar en las aplicaciones que más usuarios potenciales tengan y, como es lógico, éstas son las que están en los puestos más altos de los ranking de la tienda de aplicaciones del terminal.

En la siguiente gráfica se puede ver una comparación de los dos paradigmas anteriores (CPM+CPC frente a CPM) y como se diferencian ambos teniendo uno un comportamiento lineal, mientras el otro exponencial.

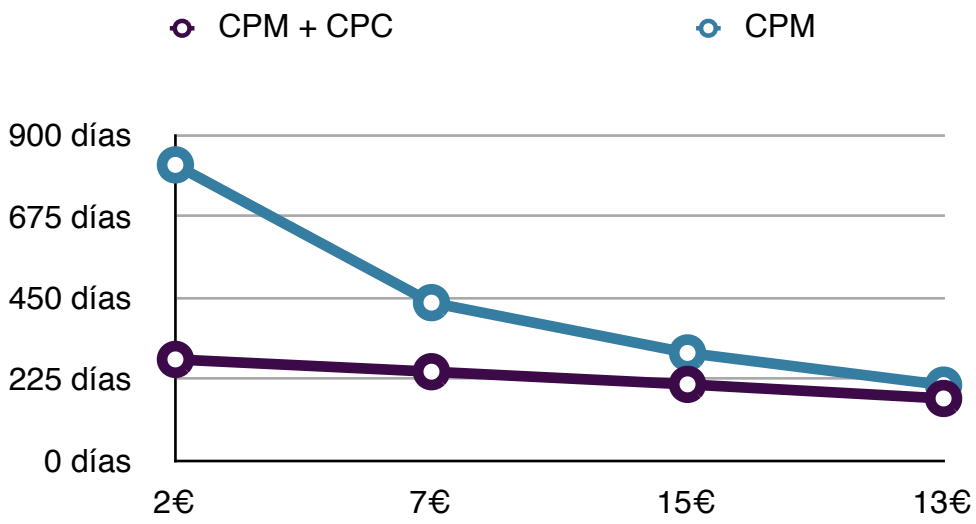

Figura 54: Comparación de CPM + CPC frente a CPM

Aún siendo el mercado de la publicidad móvil muy llamativo y prometedor, se elegirá el método tradicional de venta no gratuita debido, entre otras, a las siguientes razones:

1. El mercado al que va dirigida la aplicación, como se comentó anteriormente, es el mercado con carácter más adquisitivo en las plataformas móviles a nivel mundial, cosa que se debe explotar.

2. Para implementar iAd, habría que portar la aplicación para la versión 4.0. Lo que se traduce en más tiempo invertido y menos dinero recaudado.

3. El modelo iAd todavía está inmaduro. En el momento en que se están escribiendo estas líneas, ni siquiera está disponible. Es preferible ver como evoluciona el mercado, comprobar que las estimaciones efectuadas son acertadas y actuar en consecuencia. Cabe destacar que siempre se puede volver atrás y cambiar el modelo si se prevé que va a ser más rentable la otra opción.

Capítulo 6: Conclusiones y líneas futuras

En este capítulo se exponen las conclusiones en relación al proyecto así como las distintas líneas de futuro para continuar el mismo.

6.1 Conclusiones sobre el proyecto

Cuando se planteó la idea principal del proyecto, se pensó en una aplicación orientada exclusivamente al mercado turístico. Una aplicación que simplemente mostrara lugares con tal carácter que estuvieran cercanos al usuario y estableciera diversas rutas entre los mismos. Mirando ese planteamiento inicial de hace aproximadamente 9 meses y viendo el resultado final, las conclusiones del proyecto no pueden ser otras sino muy positivas y satisfactorias.

Como se ha mencionado, lo que en un principio trató de ser una aplicación de rutas entre lugares turísticos cercanos, ha acabado convirtiéndose en un reemplazo completo y mejorado de la aplicación de mapas del terminal. Ello se ha conseguido expandiendo las búsquedas no sólo a lugares turísticos, sino a cualquier tipo de lugar.

El principal valor añadido de la aplicación es, sin duda, la posibilidad de ver fotos, vídeos y opiniones de los diferentes lugares desde la misma aplicación y mediante una interfaz simple y acorde a los estándares nativos de la plataforma. Dichas funcionalidades se pensaron teniendo siempre en mente un mercado turístico y lo que los usuarios de tal ámbito desean ver y conocer antes de visitar un lugar.

La decisión de abrir la aplicación hacia todo tipo de sitios viene dada por dos motivos principales:

1. La posibilidad de ver fotos, vídeos y opiniones que brinda el software, aunque recibe una importancia especial para lugares turísticos, es de gran relevancia también para un lugar genérico y supone un añadido de agradecer en la mayoría de las ocasiones.

2. Es una forma de ampliar considerablemente el número de compras de la aplicación. Los usuarios no sólo tendrán una aplicación turística, sino una aplicación con características muy similares al servicio web Google Maps. Esto abre considerablemente el abanico de usuarios potenciales del software. Por tanto se trata de una razón de marketing.

Destacar el aspecto social con el que se ha dotado a la aplicación que, pese a ser una característica menor, es algo que el usuario estándar de hoy en día valora mucho. Las APIs utilizadas para ello han sido las de Facebook y Twitter.

La otra característica importante es la posibilidad de establecer rutas entre diferentes lugares. Existen muchas aplicaciones para calcular la ruta desde la posición actual hasta un sitio en concreto, como por ejemplo la aplicación de mapas incluida en el SO del terminal. El aspecto novedoso que se ha implementado en el proyecto es la posibilidad de establecer rutas entre más de dos lugares. Se ha conseguido de forma muy satisfactoria a través del API del servicio *CloudMade*. Como se mencionó en el capítulo tres, uno de los principales objetivos en este ámbito era conseguir mostrar la ruta dentro de la aplicación. Este requisito venía dado por la carencia de multitarea en el terminal.

Dos características no planificadas inicialmente y que hacen que la aplicación tenga una funcionalidad muy extensa y completa han sido la posibilidad de guardar los lugares favoritos del usuario y la capacidad de enviar éstos por bluetooth. Como se ve, ambas características están ligadas y supone un extra muy útil para el usuario. Para ello se ha trabajado extensamente con el API *GameKit* del SDK.

Otros aspecto a destacar es la interfaz de usuario. Como se mencionó en capítulos anteriores, este es un aspecto muy a tener en cuenta en cualquier desarrollo en plataformas Apple. Se ha requerido de un tiempo muy considerable para buscar todos los elementos gráficos de la aplicación y tratarlos con las aplicaciones correspondientes para conseguir el resultado final.

Mencionar el hecho de no haber podido implementar algo en lo que se pensó como un gran complemento de la aplicación y en lo que se dedicó cierto tiempo en investigar, esto es, dotar a la aplicación de capacidades de realidad aumentada. Finalmente se decidió que esta funcionalidad estaba fuera de los límites del proyecto y se comenta más adelante en el apartado de líneas futuras.

Como se puede ver, se ha trabajado con muchas APIs y servicios web para intentar ofrecer una aplicación que integre las capacidades y funcionalidades de Google Maps que se pueden disfrutar en la web, junto con las ventajas intrínsecas de los terminales móviles (el usuario pueda llevar la aplicación siempre consigo) y una interfaz de usuario acorde a los productos de Apple.

6.2 Conclusiones a nivel personal

Personalmente, la experiencia de realización del proyecto ha sido una de las etapas de la carrera que recordaré con más cariño. Adentrarme en el análisis, diseño e implementación de un proyecto software es algo que nunca tuve la oportunidad de hacer en toda la carrera. Ha resultado muy interesante estar presente en toda la cadena y ver

como la simple idea del principio va creciendo y desarrollándose en las distintas etapas del proyecto para conseguir llegar a un producto final de calidad.

Por otra parte, he cumplido con creces el objetivo principal por el que me adentré a realizar el proyecto, y no era otro que la curiosidad de ver como se diseñan y qué hay detrás de esas aplicaciones que día a día estaba utilizando en mi teléfono móvil. Ver que he sido capaz de desarrollar una aplicación a la altura de las mismas ha sido algo muy motivador para mí.

Por último destacar que, por necesidades intrínsecas del desarrollo del proyecto, he tenido que trabajar con un ordenador Macintosh, sustituyendo mi viejo ordenador y convirtiéndose hoy en día en mi ordenador personal que utilizo tanto para trabajar como para disfrutar de mi ocio.

6.3 Líneas futuras

En este apartado se explican las diferentes posibilidades para la continuación de las actividades realizadas en el presente proyecto. Estas posibilidades suponen una mejora en la aplicación o una adecuación al mercado y a las características demandadas por los usuarios.

6.3.1 Adecuación al nuevo hardware y software

Con la introducción del nuevo modelo de iPhone y el nuevo sistema operativo en junio de 2010, subyace una nueva serie de mejoras de la aplicación que se expondrán como líneas futuras al proyecto.

6.3.1.1 iPhone 4

La característica del nuevo terminal de Apple que es interesante de cara a la aplicación es el aumento de la resolución, que ahora llega hasta los 960 x 640 pixeles. Por tanto, sería muy conveniente aumentar la resolución de las distintas imágenes utilizadas en el proyecto para adecuarse a la nueva resolución.

6.3.1.2 iOS 4

La cuarta versión del sistema operativo de iPhone trae consigo, entre otras, las dos siguientes características relevantes para la aplicación:

- **Completado de tareas**: las aplicaciones tienen la posibilidad de terminar ciertas tareas que se han quedado incompletas cuando el usuario ha decidido cambiar a otra aplicación.

- **Cambio rápido de aplicación**: esta es la característica indispensable y que debe ser implementada por todos los desarrolladores. La aplicación entra en un estado en el cual no consume recursos del dispositivo y permite al usuario recuperar la aplicación en el estado y pantalla en el que se había quedado previamente a haber abandonado tal aplicación.

La primera de ellas puede ser utilizada dentro de la aplicación para seguir cargando fotos o vídeos de un lugar mientras la aplicación esté en segundo plano.

La segunda característica es determinante, y se deja como futura mejora de la aplicación.

Figura 55: Multitarea en iOS 4

6.3.2 Realidad aumentada

La realidad aumentada (AR - *Augmented Reality*) supone uno de los avances más importantes en la industria de la electrónica moderna. A grandes rasgos, trata de superponer gráficos, audio, o cualquier otro elemento atractivo a los sentidos, sobre un entorno de mundo real que es mostrado en una pantalla de un dispositivo (bien sea móvil o estático). Esto permite que el usuario perciba una realidad mixta a tiempo real.
Esta información puede ser de cualquier tipo, como por ejemplo la ruta necesaria para ir a un sitio de interés, el número de gente rodeando al usuario que está consultando un servicio de red social 2.0 como Facebook o incluso elementos interactivos de un videojuego (ver Figura 56).

Figura 56: Ejemplo de realidad aumentada[23]

Hay que destacar que la realidad aumentada no es un concepto nuevo. Desde hace tiempo se usa en retransmisiones deportivas, donde por ejemplo una línea recta plasmada sobre la televisión indica la distancia que existe desde el jugador hasta la portería en el momento de tirar una falta. Incluso las cámaras "*slow motion*" para ver ciertas escenas a cámara súper lenta pueden considerarse como integrantes de esta tecnología (al transformar la realidad de cierta manera para ofrecer información relevante al usuario). Pero es ahora, debido a la utilización de hardware y software de última generación en los teléfonos móviles, cuando se ha conseguido la mayor expansión de dicha tecnología.

La mayoría de las aplicaciones de realidad aumentada para dispositivos móviles se basan en el mismo sistema, esto es, superponer información de utilidad para el usuario sobre la imagen que está captando el dispositivo a través de la cámara integrada. Esta información puede ser muy diversa, pero es necesario que el terminal tenga cierto conocimiento del entorno que lo rodea. Para ello se suele basar en el cálculo de la posición del usuario (mediante las tres técnicas presentadas en el capítulo tres). Otros sistemas muy útiles y utilizados en sistemas AR son:

- **Brújula**: Para saber la orientación del usuario y el sentido al que está mirando.

- **Acelerómetro**: Detecta el movimiento y el giro del usuario.

- **Giroscopio**: Para medir rotaciones del usuario sobre el mismo eje.

[23] Imagen obtenida de [VII]

En la aplicación desarrollada en el presente proyecto, la idea más clara de aplicación de dicha tecnología sería superponer sobre la imagen de la cámara los lugares obtenidos para que el usuario pueda hacerse una idea de la posición en el espacio de cada lugar. Para clarificar un poco la idea, véase la Figura 57 donde los primeros iPhone muestran dos aplicaciones conocidas de realidad aumentada, para dar paso al prototipo que se ha diseñado (basado en los dos modelos anteriores) aplicando la realidad aumentada en el software desarrollado.

Figura 57: Realidad aumentada aplicada al proyecto[24]

6.3.3 iPad/Android

Otra opción sería portar la aplicación para que pueda ejecutarse tanto en el reciente *tablet* de Apple (iPad) como en los dispositivos con sistema operativo Android.

Destacar que la aplicación puede ya funcionar de forma nativa y sin problemas en el iPad, aunque a una resolución aumentada y pixelada. Por ello se trataría de rediseñar la aplicación para aprovechar las capacidades de una pantalla de 9.7”, lo que supondría un cambio relativamente importante en cuanto a interfaz de usuario. En lo relativo a la funcionalidad, sería prácticamente la misma que en su versión de iPhone cubierta en el proyecto.

[24] Imagenes obtenidas de VIII

Como se comentó en capítulos anteriores, Android es la otra tecnología que supone una gran cuota de mercado y sería la primera plataforma móvil a la que convendría portar la aplicación. Por tanto, como línea de futuro sería muy interesante la reescritura del código para que pueda ejecutarse en plataformas Android. Ello supone un cambio sustancial tanto en diseño de interfaz como en código. Aún así, en vistas del gran incremento en cuota de mercado que está sufriendo dicha familia de terminales, puede ser muy beneficiosa y conveniente (económicamente hablando) esta línea de futuro.

Figura 58: Aplicación para iPad y Android

6.3.4 iAd

Ya se comentó en el capítulo dos todo lo relacionado con el nuevo sistema publicitario de Apple, también conocido como iAd. Así mismo se justificó la elección del sistema de venta tradicional frente a ofrecer la aplicación de forma gratuita, consiguiendo ingresos a través de la publicidad. Entre otras razones, se defendió el hecho de que el sistema es todavía muy nuevo e inmaduro, por lo que sería conveniente dar tiempo y ver como evoluciona el mercado en este ámbito. Por tanto si, como se prevé, iAd será todo un éxito, convendrá incorporar publicidad a la aplicación para poder ofrecerla gratuitamente en un futuro e incrementar el retorno de la inversión.

Figura 59: iAd[25]

[25] Imagen obtenida de IX

Anexo A: Manual de la aplicación

Este anexo es de vital importancia ya que muestra el carácter eminentemente práctico de la aplicación desarrollada. Dicho anexo se divide en dos partes:

- Explicación de las pantallas principales, donde se detallará la función de cada botón y las partes más importantes del software.
- Usos típicos de la aplicación. En concreto se tratarán dos: búsqueda de los lugares turísticos cercanos y la planificación de un viaje.

A.1 Explicación de las pantallas principales

A.1.1 Introducción y barra superior

En el momento de arranque de la aplicación, se puede ver una pantalla principal (Figura 60 a), también conocida por el nombre de "*splash*") la cual tiene doble cometido: cargar todos los datos necesarios para la ejecución del programa y dar a conocer el nombre de la aplicación y una primera impresión de la misma. Una vez terminada la carga, el usuario se encuentra con la Figura 60 b) donde se da la posibilidad de elegir entre 5 idiomas (español, inglés, alemán, francés e italiano). En la parte superior derecha de la pantalla, se puede ver un botón de información, el cual lleva a la Figura 60 c). Una vez elegido el idioma deseado, y después de pulsar sobre el botón marcado con un "tick", se pasaría (después de una breve carga) a la Figura 61.

Figura 60: a) "*Splash*" b) Elección de idioma c)Información

Figura 61: Lista principal de lugares

En la figura 61 se puede ver una lista de los lugares (ocho en concreto). Estos podrán ser turísticos, o cualquier lugar concreto dependiendo del modo de uso que se elija. De cada lugar se puede observar su foto, el nombre, la puntuación y la distancia a la que se encuentra de la posición del terminal móvil. Se puede observar la presencia de una última celda cuyo cometido es obtener nuevos lugares (ocho más).

En el caso de que la aplicación no pueda obtener ningún lugar, el resultado sería el de la figura 62.

Figura 62: Ningún lugar

Tanto en la figura 61 como en la figura 62 se puede observar una barra superior con seis botones. En la Figura 63 se puede ver una explicación de la función de cada uno de dichos botones.

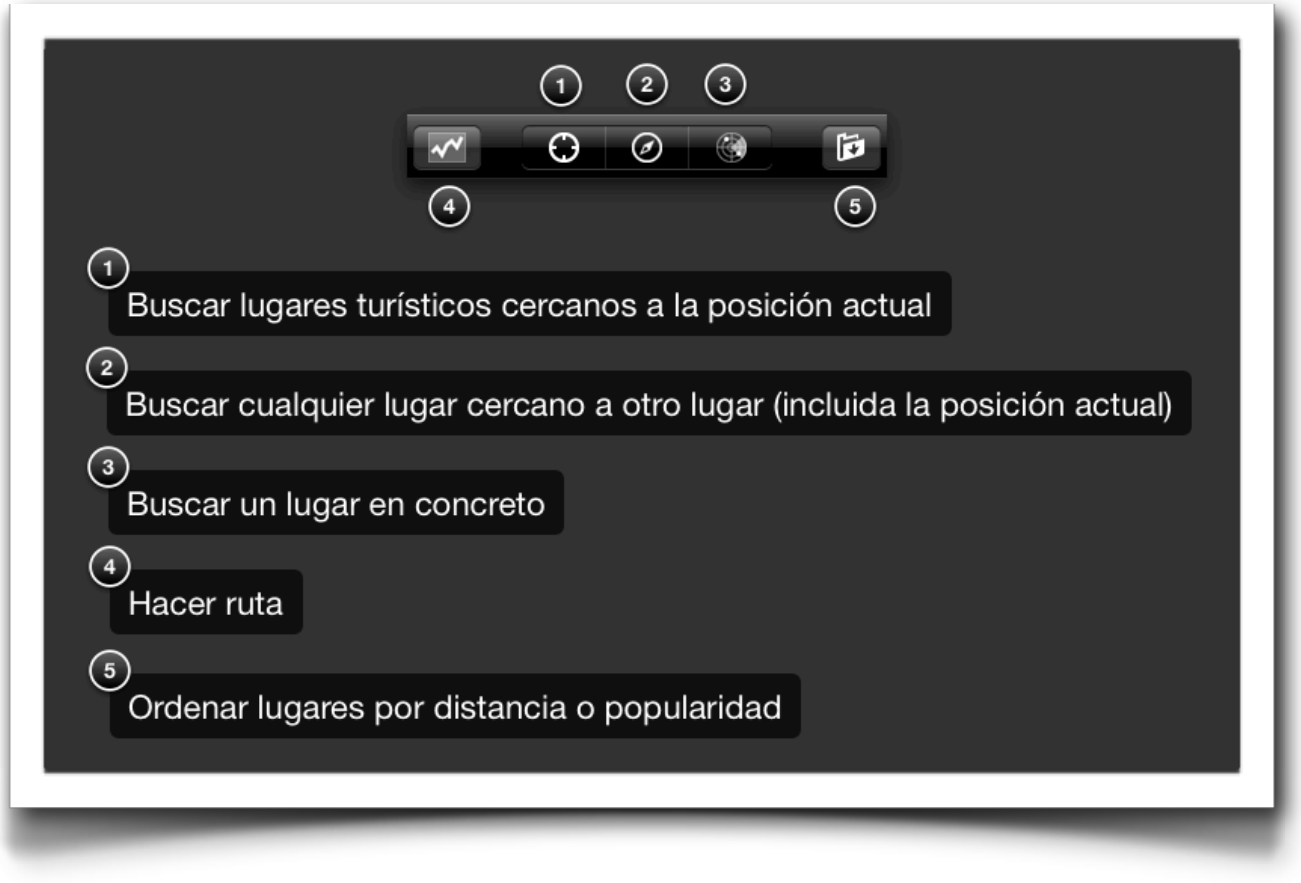

Figura 63: Explicación barra superior

Se puede observar como, debido al gran número de funciones que realiza el programa, y pese a intentar utilizar iconos que resulten autoexplicativos, un usuario corriente necesitará algo de tiempo para hacerse con la aplicación y resultar lo más cómoda posible en su utilización.

A continuación se detallarán las pantallas que surgen en cada uno de los momentos que el usuario presiona los botones detallados en la figura anterior.

En la Figura 64 se pueden ver los momentos inmediatamente posteriores a cuando el usuario pulsa los botones 2 y 3. Se puede ver como en a), la pestaña donde se introduciría el lugar tiene el valor “Posición actual”, permitiendo al usuario buscar

cualquier lugar cerca de la posición del usuario (por defecto) o cerca de cualquier otro sitio o lugar.

Figura 64: Aspecto de la aplicación a) Tras pulsar 2 b) Tras pulsar 3

En la Figura 65 se pueden ver las pantalla de selección de ruta y de ordenación de resultados tanto por distancia como por puntuación.

Figura 65: Aspecto de la aplicación a) Tras pulsar 4 b) Tras pulsar 5

Una vez explicadas las funciones de la barra superior, se pasará a explicar la estructura principal de la aplicación que reside sobre las pestañas de la barra inferior (Figura 66).

Figura 66: Explicación barra inferior

La aplicación se estructura en cuatro pestañas de las cuales cabe destacar la pestaña lista y la pestaña mapa. Antes de pasar a ellas, se verán más brevemente las pestañas favoritos y preferencias (Figura 67 a) y b) respectivamente).

A.1.2 Pestañas Favoritos y Preferencias

En la siguiente figura se puede ver el aspecto de cada una de las pestañas.

Figura 67: a) Favoritos b) Preferencias

Mediante la pestaña de ajustes se pueden cambiar la cuenta de Twitter y el idioma de la aplicación (a través de la misma pantalla que se ofrece la primera vez que se ejecuta el programa). La pestaña favoritos cumple la función de almacenar aquellos lugares relevantes para el usuario y de los que se quiera una consulta rápida posterior sin tener que pasar por los pasos usuales para conseguir los mismos. Se puede observar también en la pestaña un indicador (o "*badge*") con el número de favoritos almacenados en ese momento. La barra superior de esta pantalla se puede observar en la siguiente figura.

Figura 68: Explicación barra superior favoritos

Destacar como la función de realización de rutas sigue estando presente. Cabe mencionar la importancia de esta función en esta pantalla, y no es otra que la realización de rutas entre dos puntos cualesquiera. Anteriormente en la pestaña "Lista", se podían realizar rutas entres los lugares resultantes de una búsqueda, mientras que en la pestaña favoritos se puede saber la ruta desde un sitio A obtenido en la búsqueda X hasta un sitio B obtenido en la búsqueda Y.

También se dispone de un botón para borrar los distintos favoritos y de otro botón de envío por bluetooth para transmitir hacia otro iPhone los favoritos que se deseen de entre los que se hayan almacenado en la vida de la aplicación. En la figura 69 se puede ver la pantalla que resulta al pulsar este botón.

Figura 69: Envío de favoritos por bluetooth

Se puede observar como tras una pequeña búsqueda, se obtienen todos los dispositivos que estén corriendo la aplicación (y que hayan pulsado el citado botón) mostrándose en una lista de la cuál sólo se puede elegir uno. En cuanto a favoritos, destacar por último la posibilidad de gestionar los mismos pudiendo borrar los que ya no sean de utilidad, y la pantalla resultante cuando no se ha guardado todavía ninguno.

Figura 70: a) Borrar favoritos b) Ningún favorito

A.1.3 Pestañas Lista y Mapa

Ahora se pasará a explicar las pestañas más importantes de la aplicación, esto es, la pestaña “lista” y la pestaña “mapa”. En la primera, como su propio nombre indica, y como se ha podido ver en todas las capturas anteriores, se muestran los lugares uno debajo de otro de forma clara y concisa. La pestaña mapa da una panorámica de la situación geográfica de los lugares, comportándose de forma dual con la pestaña lista.

Figura 71: a) Lista b) Mapa

Se puede ver como de cada lugar se proporciona la siguiente información:

- Nombre
- Foto
- Distancia[26] con respecto a la posición del terminal
- Puntuación media de los usuarios

[26] Notar que las distancias obtenidas en la imagen de arriba corresponden a la distancia que existe entre los lugares mostrados y la localización del terminal que, al ser en este caso el simulador, se trata de la sede central de Apple en Cupertino, California

También mencionar los colores de los pines que representan cada uno de los lugares en el mapa. Estos colores dependen de la puntuación de cada lugar y dan una panorámica visual muy rápida de la importancia turística del lugar que se esté tratando:

- Pin rojo: más de cuatro estrellas
- Pin verde: entre dos y cuatro estrellas.
- Pin violeta: menos de dos estrellas.

Se puede ver que la barra superior en la vista de mapa tiene unas ligeras diferencias que se explicarán a continuación en la figura siguiente.

Figura 72: Barra superior mapa

El botón “1” es el equivalente a la celda “Cargar más lugares” mostrada en la Figura 61. Tras pulsar en él se vería como en el mapa caerían nuevos pines correspondientes nuevos lugares. El botón “2” es exclusivo en este tipo de vista e indica la posición actual del usuario mediante un punto azul en el mapa como se verá más adelante.

A.1.4 Pantalla principal

Una vez explicadas las dos pantallas que muestran los diferentes lugares, se pasará a analizar la pantalla principal de cada lugar. Para acceder a ella desde la pestaña “Lista” sólo se deberá pulsar en la celda correspondiente al lugar deseado. Por el contrario, para acceder a ella desde la pestaña “Mapa” se tendrá que pulsar en el pin deseado, surgiendo una especie de burbuja con la foto, el nombre, y la calle del lugar como se puede ver en el mapa de la Figura 71. Pulsando en la flecha azul de dicha burbuja, daría acceso a la citada pantalla mostrada abajo.

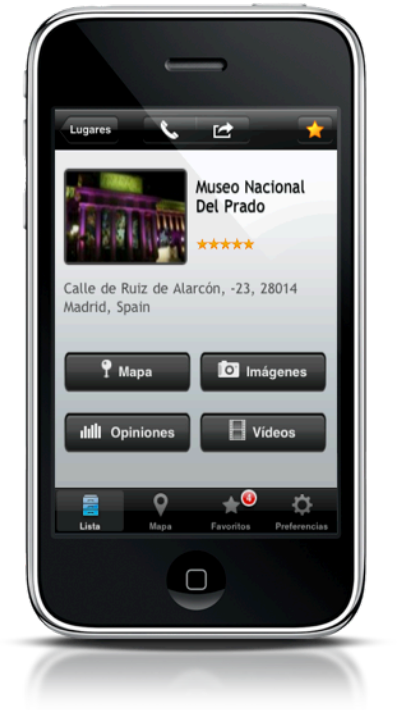

Figura 73: Pantalla principal de un lugar

Esta pantalla tiene un gran número de opciones que se tratarán de explicar a continuación. En un principio, se puede ver como las cuatro acciones principales que se tienen sobre un lugar son:

- Ver su posición en el mapa.
- Obtener sus imágenes.
- Leer sus opiniones.
- Ver sus vídeos.

Ahora se verán las pantallas resultantes de pulsar en cada uno de los cuatro botones anteriores.

A.1.4.1 Posición en el mapa

En la Figura 74 se puede ver como se representa en el mapa el lugar indicado. A diferencia de cuando se representan varios lugares, esta vez se representa la imagen del mismo para mejorar la experiencia del usuario. Esta pantalla resulta importante a la hora de realizar rutas desde la posición actual del usuario hasta el lugar representado como se verá posteriormente.

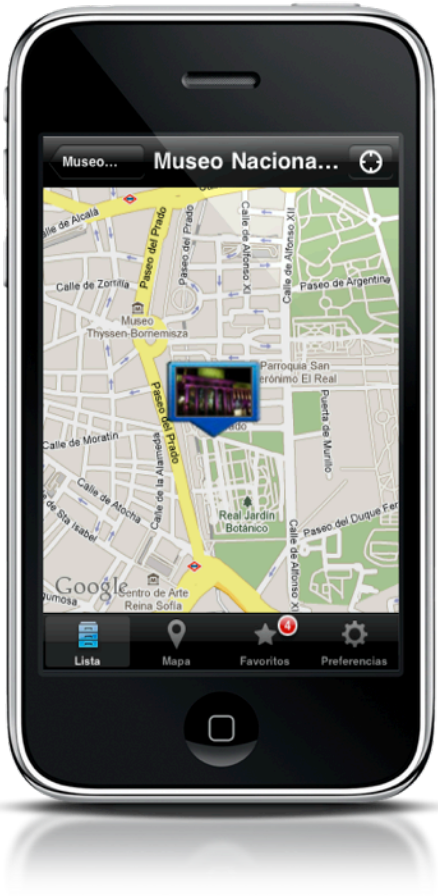

Figura 74: Mapa del lugar

Las tres siguientes características suponen las características más diferenciadoras de la aplicación y son el gran valor añadido que se tiene con respecto a las competidoras en el mismo mercado.

A.1.4.2 Imágenes

La primera de ellas es la posibilidad de ver las fotos del lugar como se puede ver en la siguiente figura. Esto supone, sin lugar a dudas, el principal atractivo de cara al usuario.

Figura 75: Fotos del lugar

A.1.4.3 Opiniones

La segunda característica son las opiniones que, al tratarse de lugares turísticos, tienen una importancia especial. En cada opinión se puede ver una pequeña parte de la misma, acompañada de la página web de donde proviene. En el caso de que se desee leer la opinión completa, simplemente se pincharía en esta última para acceder a dicha página web directamente desde la propia aplicación sin tener que salir y utilizar el navegador por defecto del teléfono.

Figura 76: Opiniones del lugar

A.1.4.4 Vídeos

Aunque importante en menor medida que las anteriores posibilidades, los vídeos suponen un gran extra para el usuario que potencia la experiencia de usuario de la aplicación. Notar el hecho de que los vídeos son cargados directamente por la aplicación, sin tener que salir de la misma para cargar la aplicación “Youtube” propia del teléfono.

Figura 77: Vídeos del lugar

A.1.4.5 Otras opciones

Una vez analizadas las opciones más importantes que se tienen disponibles para cada lugar, se verán otras opciones disponibles a través de la barra superior de la imagen de la Figura 78.

Figura 78: Barra superior pantalla principal de un lugar

El botón 3 sirve para almacenar el lugar en la pestaña favoritos para tenerlo disponible siempre que se desee. Tras pulsar el botón “1”, se obtiene el menú desplegable mostrado en la Figura 79, teniendo la posibilidad de llamar (no disponible en iPod Touch) o de guardar el contacto.

Figura 79: Llamar o agregar lugar

Tras pulsar el botón “2” se la posibilidad de compartir el lugar en cuestión bien por email, o por medio de las redes sociales Facebook y Twitter. Está característica es cada vez más demandada por el usuario e impregna la aplicación con un tinte 2.0 tan necesario en el software actual.

En la Figura 80 se puede ver lo sencillo que resulta publicar algo en el tablón de Facebook. Notar que la pantalla para introducir el usuario y la contraseña sólo aparecerá la primera vez que se realice dicha acción.

Figura 80: Recomendar lugar (I)

A continuación se muestran las dos opciones restantes, esto es, envío a través del correo electrónico y publicación en el tablón de Twitter.

Figura 81: Recomendar lugar (II)

Una vez analizadas las distintas pantallas y grandes posibilidades que tiene la aplicación, se pasará a ver alguno de los usos más típicos que tendría.

A.2 Usos típicos de la aplicación

En este apartado se tratará de mostrar los usos eminentemente prácticos que posibilita la aplicación desarrollada, mostrando especial interés por algunas características de la aplicación no mostradas en el apartado anterior como son la realización de rutas.

A.2.1 Búsqueda de lugares cercanos

La siguiente sección se divide en dos tipos de lugares: *turísticos* y *no turísticos*.

A.2.1.1 Lugares turísticos

Esta es la característica principal de la aplicación y la meta con la que se desarrolló el proyecto. Nada más arrancar el programa (o cada vez que se pulse en el botón 1 de la Figura 63 en la pestaña “Lista” o en su equivalente en la pestaña “Mapa”) se obtendrán una serie de sitios turísticos alrededor del lugar donde se encuentra el usuario. Destacar que el lugar elegido para realizar las simulaciones ha sido la Universidad Carlos III en Leganés. Imaginando por tanto que se quieren buscar lugares atractivos, turísticamente hablando, cercanos al lugar mencionado, el resultado de la aplicación (una vez aplicado el filtro de popularidad) sería:

Figura 82: Lugares turísticos cercanos

En la vista de mapa se puede ver una panorámica de los diferentes sitios (categorizados por sus pines de colores) rodeando a la posición actual del usuario. Esto da una idea de lo turística que es la zona con un simple vistazo a la pantalla. Supóngase que, tras acceder a la pantalla principal del lugar elegido (en este caso “Teatro La Latina”), las diferentes imágenes, opiniones y vídeos del mismo despiertan claramente en el usuario el interés necesario como para visitarlo. Aquí es donde entra en juego la vista de mapa con la inclusión de las rutas como se puede ver a continuación.

Figura 83: Ruta desde la posición actual

Se puede observar como la aplicación ofrece dos tipos de ruta, esto es, en coche o a pie. En el caso de la Figura 83 se eligió el primero de los modos y el resultado de la ruta se puede ver directamente desde la misma aplicación, pudiendo optar por salir de la misma para verse en la aplicación “Mapas” del propio iPhone como se ve en la Figura 84.

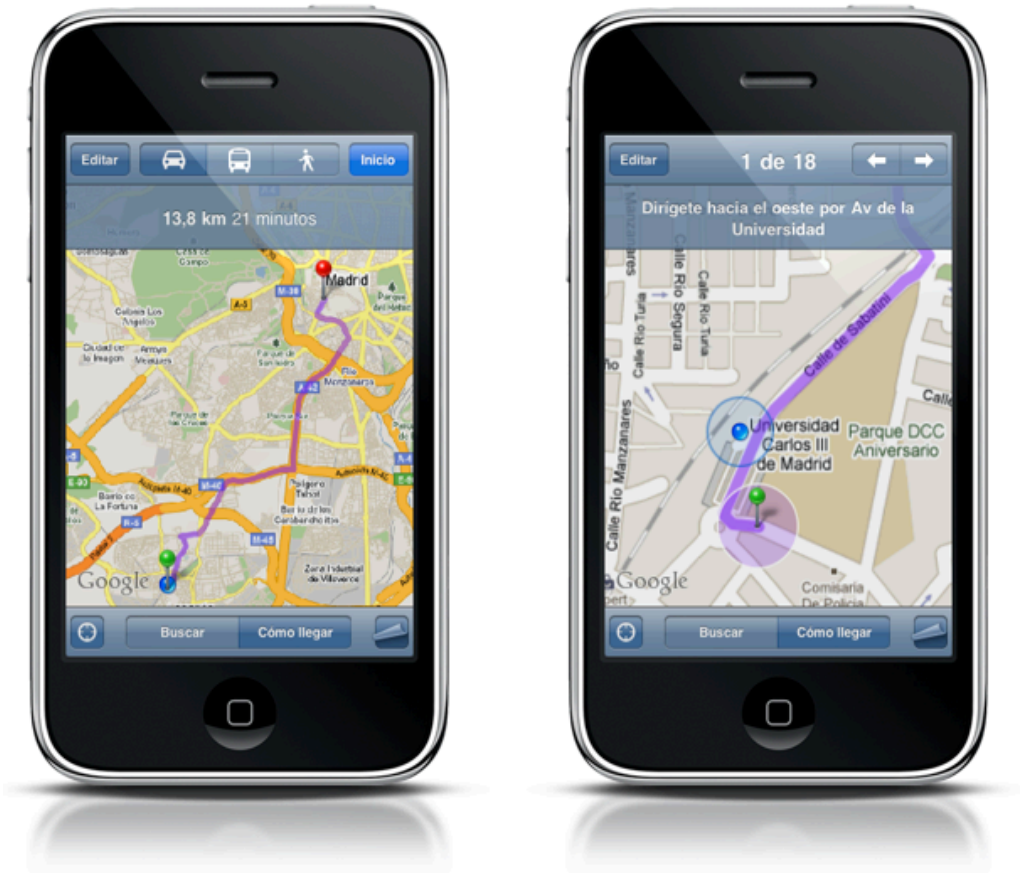

Figura 84: Ruta en la aplicación “Mapas”

También destacar dos opciones muy interesantes implementadas directamente en la aplicación y que se pueden ver en la Figura 85:

- Posibilidad de ver las indicaciones directamente desde la propia aplicación. Aunque muchas veces es suficiente el hecho de ver la ruta dibujada, nunca está de más poder ver las indicaciones paso a paso (*“Turn-by-Turn Navigation Systems”*).

- Posibilidad de guardar la imagen de la ruta en la fototeca del iPhone. Esto resulta de especial interés en rutas cortas donde se puedan apreciar las distintas calles o para diferentes zonas de una ruta larga ampliando la zona en cuestión.

Figura 85: Barra superior ruta (I)

Se pasará ahora a explicar una opción que aporta gran valor añadido a la aplicación, y es la posibilidad de hacer rutas entre diferentes puntos turísticos cercanos al usuario. Como se explicó en la Figura 63, la barra superior en la vista “Lista” dispone de un botón con el que se accede al modo de edición de ruta y donde se pueden elegir cada uno de los lugares que se desean incluir en la misma (incluida la posición actual).

Figura 86: Ruta con varios lugares

Se puede ver como de nuevo se disponen las indicaciones paso a paso mostradas anteriormente y, como novedad, se dispone de dos nuevas formas de representación del mapa, como son la vista híbrida y la vista satélite.

Figura 87: Indicaciones paso a paso y vista satélite

Otra característica novedosa que se ha añadido, y que supone una opción muy interesante, es la de tener la posibilidad de ver esa ruta en la versión web de Google Maps, con todo lo que ello implica (por ejemplo la inclusión de *Panoramio*).

Figura 88: Barra superior ruta (II)

Figura 89: Ruta en Google Maps

A.2.1.2 Lugares no turísticos

Para terminar esta sección cabe mencionar el hecho de que, aunque en un principio la aplicación fue diseñada con una fuerte orientación hacia el mercado y los lugares de carácter turístico, puede ser utilizada para buscar cualquier tipo de lugar cercano. Supóngase que en la ruta anteriormente mostrada, la entrada para el primer sitio de la misma ("Teatro Auditorio Federico García Lorca") es más cara de lo que se suponía y se necesita encontrar un cajero cercano para solventar el problema. La aplicación permite, una vez calculada la posición actual utilizada para proporcionar al usuario los lugares turísticos en los alrededores, buscar cualquier tipo de lugar en la zona. Destacar que para esta clase de lugares, no se obtendrán ni puntuaciones, ni comentarios, ni fotos ni vídeos (en la mayor parte de los casos), pero se conseguirá lo que se buscaba, es decir, cómo llegar al lugar cercano.

Figura 90: Búsqueda de lugares no turísticos

A.2.2 Planificación de un viaje

A continuación se explicará brevemente un caso práctico en el que la aplicación desarrollada sería de gran ayuda, como es la organización y planificación de un viaje.

Supóngase que se desea realizar un viaje a París. Se requieren, en un primer momento, dos cosas:

- El hotel

- Atracciones turísticas correspondientes en las que se va a invertir el valioso tiempo del viaje.

Ambas cosas se pueden buscar de manera muy sencilla y cómoda con el teléfono móvil y la aplicación. Se empezará por buscar en un primer momento el hotel que, supóngase, se desea que se encuentre lo más próximo posible a la Torre Eiffel.

Figura 91: Búsqueda de hotel

Mediante la posibilidad que da la aplicación de ver la situación geográfica, las opiniones, y las fotografías del lugar, es muy sencillo poder elegir y descartar entre las distintas opciones. Tras haber elegido el hotel deseado (en este caso “Hotel Tour Eiffel”) se deberá agregar a favoritos para poder realizar las acciones que se verán a posteriori.

Figura 92: Elección de hotel

Tras la elección del hotel, sólo resta buscar los sitios turísticos deseados (en este caso museos) y agregarlos a favoritos para poder tener la posibilidad de tener la ruta desde el hotel hasta los diferentes sitios turísticos elegidos. La búsqueda se ha realizado utilizando dos de las características que ofrece la aplicación:

- Buscando los museos cercanos a la Torre Eiffel (botón 2, Figura 63)
- Buscando el Museo del Louvre en concreto (botón 3, Figura 63)

Una vez elegidos los lugares, se deberán marcar como favoritos para poder realizar la buscada ruta desde el hotel. La Figura 93 muestra como se ha conseguido la misma y, por tanto, conseguir planificar un futuro viaje desde el propio teléfono móvil gracias a la aplicación.

Figura 93: Ruta del viaje planificado

Anexo B: Backup y control de versiones

Desde un primer momento se trató de buscar la mejor manera de tener un tanto un backup como un control de versiones adecuado para tener el proyecto seguro y controlado en todo momento. En este anexo se explicarán las herramientas utilizadas para conseguir estos dos requisitos.

B.1 Backup - Dropbox

Debido a la no tan despreciable probabilidad de fallo de los discos externos y existiendo la posibilidad de realizar un backup online gratuito, ya que no se requiere gran cantidad de espacio, se eligió esta última opción. Existen numerosas herramientas en Internet pero se ha utilizado Dropbox (www.dropbox.com) por ser extremadamente simple, gratuito y con una serie de ventajas que se tratarán de detallar brevemente a continuación.

La mayoría de los servicios de backup online se limitan a ofrecer un espacio de almacenamiento en la nube que puedes gestionar vía FTP o un navegador web. La clave de Dropbox radica en la existencia de un cliente local sincronizado con el servidor de almacenamiento online. El funcionamiento de Dropbox es muy simple, por un lado se dispone de una carpeta en nuestro ordenador, y por el otro una interfaz web para acceder a los archivos. Cualquier fichero que se introduzca en la carpeta será automáticamente subido a la “nube”. De igual forma, cualquier fichero que se suba mediante la interfaz web de Dropbox, será automáticamente bajado a la carpeta local del ordenador. Por tanto se tiene una carpeta del sistema sincronizada en todo momento con el servidor online en cuestión y cualquier cambio que se haga en cualquiera de las partes será reflejado inmediatamente en la otra.

La otra ventaja clave para utilizarlo como backup es la posibilidad que ofrece Dropbox de volver a versiones antiguas de cada archivo. Esto se consigue de forma más flexible con el control de versiones del que se hablará más adelante en el anexo pero, aún así, es una característica muy importante a tener en cuenta.

Dropbox está disponible en las tres plataformas (Windows, Mac y Linux) y dispone de un programa tanto para iPhone como para Android, lo que permite tener acceso a los archivos en cualquier lugar y supone una gran diferenciación sobre otros sistemas similares de backup que no disponen de la multicanalidad que ofrece Dropbox.

Figura 94: Dropbox multiplataforma[27]

Destacar el hecho de poder compartir archivos y carpetas fácilmente con otro usuario de Dropbox, de tal forma que la carpeta compartida por ambos será sincronizada entre las dos cuentas. Este ha sido el método elegido para intercambiar toda la información referente al proyecto con el tutor del mismo.

Volviendo a la cuestión anterior, la principal razón por la que Dropbox es el servicio más popular y utilizado hoy en día es, en gran medida, por el hecho de disponer del cliente local. Esto resulta de especial interés cuando se usan diferentes ordenadores, y se quieren tener todos los programas sincronizados entre ellos. La clave radica en indicar a las distintas aplicaciones que su carpeta con la información específica no reside en el sitio por defecto, sino en una carpeta dentro de Dropbox. Esta carpeta estará por tanto sincronizada con Dropbox y con los distintos ordenadores enlazados con la cuenta que se use. De esta forma se pueden conseguir posibilidades que van más allá de la básica sincronización de archivos.

Más allá de dichas posibilidades, el uso que se ha dado a dicho servicio en la realización del proyecto ha sido como mero servicio de backup y de compartición de ficheros entre dos ordenadores (en concreto el personal y el del trabajo) y con el tutor.

[27] Imagen obtenida de http://www.error500.net/images/articulos/dropbox.jpg

B.2 Control de versiones - Git

B.2.1 Descripción del servicio

En cuanto al método de control de versiones elegido, éste ha sido *Git* debido a varias ventajas sobre *subversion*:

- Sistema distribuido (al clonar el repositorio, se obtiene una copia de toda la historia del proyecto y no sólo la última versión como ocurre en *subversion*).
- Se trata de un paradigma completamente nuevo debido a la ramificación ("*branching*").
- Potencia la creatividad y la originalidad.
- Gran rendimiento debido a estar escrito en C.

Donde destaca Git con respecto a los demás sistemas de control de versiones es en la incorporación de un nuevo paradigma basado en la ramificación. Una rama (*branch*) actúa de igual manera que un borrador en el gestor de correo. Se trabaja en este borrador, guardando frecuentemente hasta que se termina, se envía y se elimina.

La filosofía de trabajo en Git es la siguiente: existe una rama maestra (*master*) que se supone que es la última versión del proyecto limpio y perfectamente operativo. Cuando se quiere implementar una nueva funcionalidad, se crea una rama. Esto automáticamente hace que cambie el sistema de ficheros en el que se estaba trabajando y cualquier cambio que se efectúe a continuación sólo tenga efecto en esta rama, de tal forma que la rama maestra jamás sufrirá ninguna modificación estando perfectamente a salvo. Una vez se ha terminado de desarrollar la característica por la que se creo tal rama (y se han pasado los tests correspondientes), ésta se fusiona con la rama maestra borrándose la anterior.

Git está cada día obteniendo más cuota de mercado y desbancando a subversión en muchas empresas (sobre todo empresas modernas y de nueva creación).

Por tanto, la solución que se ha elegido en el proyecto para tener los datos a salvo y realizar un control de versiones adecuado ha sido combinar estas dos novedosas herramientas gratuitas.

B.2.2 Configuración del control de versiones

Ahora se pasará a explicar en detalle como se ha configurado el repositorio Git en Dropbox. Se indicará paso a paso los comandos que se deben introducir en la terminal para configurar correctamente el sistema implementado. Para intentar explicar el flujo de

trabajo elegido (ordenador en casa y en el trabajo), supóngase que los siguientes comandos han sido creados en el ordenador de casa y que el nombre del repositorio se ha denominado del mismo modo que el nombre de la aplicación (*iTourist*).

1. Se crea el repositorio en una carpeta de Dropbox (en este caso en concreto PFC).

   # mkdir -p ~/Dropbox/PFC/iTourist.git

2. En la carpeta donde se tenga el proyecto realizado, se inicializa Git, se clona el repositorio y, por último, se indica la ruta del repositorio remoto:

   # git init
   # git clone --bare . ~/Dropbox/PFC/iTourist.git
   # git remote add dropbox ~/Dropbox/PFC/iTourist.git

Una vez configurado el repositorio en Dropbox, el flujo de trabajo siempre sería el siguiente:

Ordenador en casa

1. Se actualizan posibles cambios que hayan podido producirse en el trabajo:

   # git pull dropbox master

2. Se trabaja sobre el proyecto y, cuando se desee, se guardan los cambios en Git y se suben al servidor

   # git add .
   # git commit -m "información sobre los cambios'
   # git push dropbox master

Ordenador en el trabajo

1. Si todavía no se tiene el directorio de trabajo (porque sea la primera vez o porque se haya borrado), se clona el repositorio Git y se cambia el nombre por defecto del repositorio a *dropbox* (esto último es opcional y por defecto Git lo denomina *origin*)

   # git clone ~/Dropbox/PFC/iTourist.git
   # git remote add dropbox ~/Dropbox/PFC/iTourist.git

En los demás casos, simplemente se actualizarán los cambios.

 # git pull dropbox master

2. Al igual que antes, cuando se desee se guardan los cambios en el repositorio remoto de Git en Dropbox.

   # git add .
   # git commit -m "información sobre los cambios'
   # git push dropbox master

Destacar que, en este caso, la cuenta Dropbox a través de la cual se sincroniza el proyecto es la misma, pero supóngase el caso en el que varios desarrolladores estén trabajando en el mismo proyecto, cada uno con su cuenta Dropbox personal. El único cambio que habría que hacer a lo expuesto anteriormente sería convertir la carpeta *PFC* en una carpeta compartida por cada uno de los integrantes en el proyecto. Todos compartirían el repositorio Git y, por tanto, todos verían reflejados los cambios en local que cada uno de los desarrolladores haya llevado a cabo.

Figura 95: Sincronización entre casa y el trabajo

A continuación se explicarán ciertos casos especiales de Git que hay que tener en cuenta al realizar un proyecto de desarrollo software para iPhone y, en general, para desarrollo de aplicaciones en plataformas Apple. Debido a las características internas de los ficheros que componen tales proyectos, es necesario obviar ciertos archivos relacionados con la compilación y interfaz del programa. Esto es así debido a que, en un flujo normal de trabajo, cuando se guardan los cambios en Git (*commit*), el IDE de desarrollo (*Xcode*) no ha sido cerrado y, por tanto, habrá ficheros relacionados con el mismo que den problemas. Para tal efecto, Git dispone de un fichero llamado .gitignore. El fichero utilizado ha sido el siguiente.

```
# xcode noise
build/*
*.pbxuser
*.mode1v3
*.perspectivev3

# old skool
.svn

# osx noise
.DS_Store
profile
```

Para terminar este anexo, se mostrará una captura de pantalla del programa utilizado para gestionar el repositorio Git, llamado *GitX* (ver Figura 96). Se pueden observar las distintas ramas creadas (en este caso sólo *master*) y los diferentes *commits* dentro de la misma.

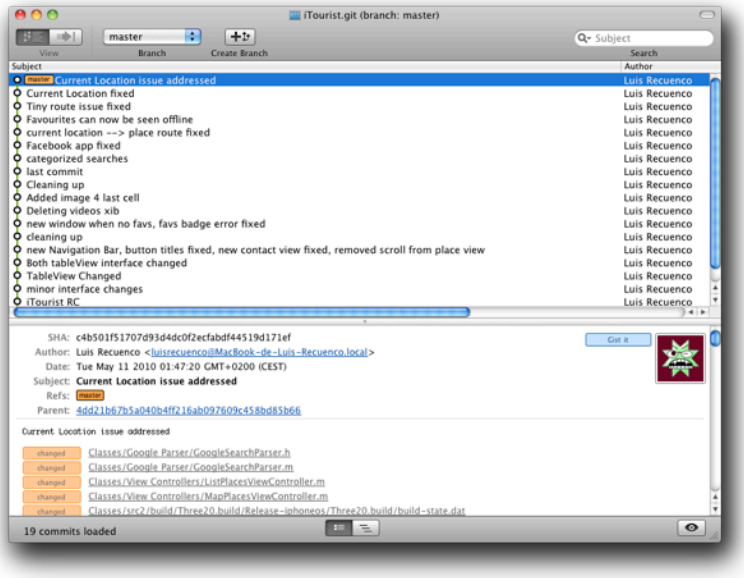

Figura 96: Gestión del repositorio Git

Fuentes de las imágenes utilizadas

[I] *Padres del email y celular ganan Príncipe de Asturias* [en línea] <http://eleconomista.com.mx/notas-online/tecnociencia/2009/06/17/padres-celular-email-_ganan-principe-asturias> [Consulta: Septiembre, 2010]

[II] *DynaTAC 8000X, el primer móvil de la historia* [en línea] <http://www.abadiadigital.com/articulo/dynatac-8000x-el-primer-movil-de-la-historia/> [Consulta: Septiembre, 2010]

[III] *SmartPhones en oferta: los costos ocultos* [en línea] <http://www.celularis.com/smartphones/smartphones-en-oferta-los-costos-ocultos.php> [Consulta: Septiembre, 2010]

[IV] Formica, Ariel. *Location Based Services* [en línea] <http://www.galileoic.org/la/files/LBS%20LOCALIZAR-T.pdf> [Consulta: Septiembre, 2010]

[V] Universidad de Stanford. iPhone Application Development [en línea] <http://www.stanford.edu/class/cs193p/cgi-bin/drupal/system/files/lectures/15_DeviceAPIs.pdf> [Consulta: Abril, 2010]

[VI] Trevisani, Emiliano; Vitaletti, Andrea. *Cell-ID location technique, limits and benefits: an experimental study* [en línea] <http://cens.ucla.edu/~mhr/cs219/location/trevisani04.pdf> [Consulta: Abril, 2010]

[VII] *Realidad Aumentada, el futuro ya está aquí* [en línea] <http://chibacity.wordpress.com/2009/03/23/realidad-aumentada-el-futuro-ya-esta-aqui> [Consulta: Agosto, 2010]

[VIII] *Layar for iPhone comes with interesting layers* [en línea] <http://www.winandmac.com/mobile/layarforiphone-review/> [Consulta: Agosto, 2010]

[IX] - Página oficial de Apple sobre iAd [en línea] <http://advertising.apple.com/> [Consulta: Septiembre, 2010]

Bibliografía

[1] Mark, Dave; LaMarche, Jeff. *Beginning iPhone 3 Development. Exploring the iPhone SDK.* Apress. 2009.

[2] Dalrymple, Mark; Knaster, Scott. *Learn Objective-C on the Mac.* Apress. 2009.

[3] Swain, Sampad. *Can Location Based Services (LBS) make Indian Internet's Growth Quicker? Part1* [en línea] <http://sampadswain.com/2009/11/can-location-based-services-lbs-make-indian-internets-growth-quicker-part-1/> [Consulta: Noviembre, 2009]

[4] Página Web sobre análisis de tiendas de aplicaciones móviles [en línea] <http://www.distimo.com/> [Consulta: Noviembre, 2009]

[5] Página Web oficial de desarrollo en Android [en línea] <http://developer.android.com/sdk/> [Consulta: Noviembre, 2009]

[6] Android Developer's Guide [en línea] <http://developer.android.com/guide/index.html> [Consulta: Noviembre, 2009]

[7] Página Web oficial de desarrollo en iPhone [en línea] <https://developer.apple.com/iphone/> [Consulta: Noviembre, 2009]

[8] KML Tutorial [en línea] <http://code.google.com/intl/en/apis/kml/documentation/kml_tut.html> [Consulta: Noviembre, 2009]

[9] KML Reference [en línea] <http://code.google.com/intl/en/apis/kml/documentation/kmlreference.html> [Consulta: Noviembre, 2009]

[10] KML Documentation Introduction [en línea] <http://code.google.com/intl/en/apis/kml/documentation/> [Consulta: Noviembre, 2009]

[11] Fielding, Roy. *Representational State Transfer (REST*) [en línea] <http://www.service-architecture.com/web-services/articles/representational_state_transfer_rest.html> [Consulta: Noviembre, 2009]

[12] Guía breve de tecnologías XML [en línea] <http://www.w3c.es/divulgacion/guiasbreves/tecnologiasxml> [Consulta: Noviembre, 2009]

[13] Extensible Markup Language (XML) [en línea]
<http://www.w3.org/XML/> [Consulta: Noviembre, 2009]

[14] JavaScript Object Notation [en línea]
<http://php.net/manual/es/book.json.php> [Consulta: Noviembre, 2009]

[15] Página Web oficial del formato JSON [en línea]
<http://www.json.org/> [Consulta: Noviembre, 209]

[16] JSON Framework [en línea]
<http://code.google.com/p/json-framework/> [Consulta: Enero, 2010]

[17] Core Location Framework [en línea]
<http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CoreLocation_Framework/CoreLocation_Framework.pdf> [Consulta: Enero, 2010]

[18] Página Web de Minube [en línea]
<http://www.minube.com/> [Consulta: Diciembre, 2009]

[19] Documentación del API de Minube [en línea]
<http://www.minube.com/api/documentation> [Consulta: Diciembre, 2009]

[20] Página Web de Qype [en línea]
<http://www.qype.es/> [Consulta: Diciembre, 2009]

[21] Página Web de 11870 [en línea]
<http://11870.com/> [Consulta: Diciembre, 2009]

[22] Documentación del API de Qype [en línea]
<http://apidocs.qype.com/> [Consulta: Diciembre, 2009]

[23] Documentación del API de 11870 [en línea]
<http://11870.com/api> [Consulta: Diciembre, 2009]

[24] Página Web de CloudMade [en línea]
<http://cloudmade.com/> [Consulta: Diciembre, 2009]

[25] Página Web del API de CloudMade [en línea]
<http://developers.cloudmade.com/> [Consulta: Diciembre, 2009 - Febrero, 2010]

[26] Página Web del API de Google Maps [en línea]
<http://code.google.com/intl/es-ES/apis/ajaxsearch/documentation/reference.html>
[Consulta: Diciembre, 2009 - Enero, 2010]

[27] Página web del Framework de Facebook para iPhone [en línea]
<http://developers.facebook.com/docs/guides/mobile/> [Consulta: Febrero, 2010]

[28] Página Web oficial del protocolo OAuth [en línea]
<http://oauth.net/> [Consulta: Mayo, 2010]

[29] *Reverse Geocoding Webservices* [en línea]
<http://www.geonames.org/export/reverse-geocoding.html> [Consulta: Enero, 2010]

[30] QuartzCore Framework Reference [en línea]
<http://developer.apple.com/library/mac/#documentation/GraphicsImaging/Reference/QuartzCoreRefCollection/index.html%23/library/mac/documentation/GraphicsImaging/Reference/QuartzCoreRefCollection/_index.html> [Consulta: Febrero, 2010]

[31] Framework *Three20* [en línea]
<http://github.com/facebook/three20/> [Consulta: Febrero, 2010]

[32] *International Standard ISO/IEC 8859-2* [en línea]
<http://webstore.iec.ch/preview/info_isoiec8859-2%7Bed1.0%7Den.pdf> [Consulta: Febrero 2010]

[33] Google Mobilizer [en línea]
<http://www.google.com/gwt/n> [Consulta: Septiembre, 2010]

[34] Instapaper Mobilizer [en línea]
<http://www.instapaper.com/m> [Consulta: Septiembre, 2010]

[35] Gamekit Framework Reference [en línea]
<http://developer.apple.com/library/ios/#documentation/GameKit/Reference/GameKit_Collection/_index.html> [Consulta: Mayo, 2010]

[36] iPhone Human Interface Guidelines [en línea]
<http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/DevelopingSoftware/DevelopingSoftware.html> [Consulta: Noviembre, 2009]

[37] Universidad de Stanford. iPhone Application Development [en línea]
<http://www.stanford.edu/class/cs193p/cgi-bin/drupal/> [Consulta: Noviembre, 2009]